POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Site Web com Inteligência Artificial

Licenciatura em Engenharia Informática

Edgar Filipe da Silva Mendes

João Rafael Freitas Oliveira

Leiria, julho de 2023

POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Site Web com Inteligência Artificial

Licenciatura em Engenharia Informática

Edgar Filipe da Silva Mendes

João Rafael Freitas Oliveira

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor João da Silva Pereira e do Professor Rui Vasco Guerra Baptista Monteiro

Leiria, julho de 2023

Agradecimentos

Gostaríamos de expressar a nossa mais sincera gratidão a ambos os nossos orientadores, Professor Doutor João da Silva Pereira e Professor Doutor Rui Vasco Monteiro, pelo apoio inestimável que nos proporcionaram ao longo da elaboração do nosso projeto informático. Sem a orientação e o conhecimento dos mesmos, certamente não teria sido possível alcançar esse marco significativo.

Agradecemos também aos nossos amigos e família pelo apoio dado ao longo da realização deste trabalho.

Resumo

Todos reconhecemos que a meteorologia e as condições climáticas têm uma influência significativa em todas as atividades humanas, quer sejam realizadas em espaços interiores ou ao ar livre. Além de influenciar as decisões mundanas de alguém antes de sair de casa, a previsão de precipitação é crucial em áreas como aviação, navegação, agricultura, indústria, comércio e turismo.

Em Portugal, o Instituto Português do Mar e da Atmosfera (IPMA), uma organização governamental, analisa e faz a previsão do estado do tempo para períodos longos até 10 dias e com um certo grau de incerteza. Por esse motivo, o objetivo deste projeto é alcançar uma alta taxa de acerto na identificação de padrões de precipitação para que o algoritmo seja confiável em contextos reais.

Para atingir esse objetivo, foram utilizados os dados do IPMA que procedeu à implementação de captura de imagens do estado meteorológico, imagens de radares que serão utilizadas para prever a precipitação nos diferentes distritos de Portugal. Neste projeto serão testadas técnicas de inteligência artificial com redes neuronais artificiais em *Deep Learning*.

Ao chegar ao fim do trabalho, foi observado uma certa falha por parte da rede neuronal na previsão de momentos com chuva intensa, devido à falta de dados deste tipo para o treino do mesmo. É ainda considerado que, caso seja possível obter mais dados do IPMA, quer sejam fornecidos pelo próprio, quer sejam obtidos pelo *developer* em períodos com chuva frequente e intensa, o algoritmo atingiria uma boa taxa de precisão.

Palavras-chave: *Deep Learning*, CNN, Meteorologia, IPMA, Redes Neuronais Artificiais, Precipitação

Abstract

We all recognize that meteorology and weather conditions have a significant influence on all human activities, whether indoors or outdoors. In addition to influencing someone's everyday decisions before leaving home, precipitation forecasting is crucial in areas such as aviation, navigation, agriculture, industry, trade, and tourism.

In Portugal, the Portuguese Institute for the Sea and Atmosphere, a governmental organization, analyses and predicts the weather conditions for longer periods of up to 10 days with a certain degree of uncertainty. For this reason, the objective of this project is to achieve a high accuracy rate in identifying precipitation patterns so that the algorithm can be reliable in real-world contexts.

To achieve this goal, data from IPMA was used, which implemented radars to capture images of the weather conditions. These images will be used to predict precipitation in Leiria. In this project, artificial neural networks in deep learning will be tested using artificial intelligence techniques.

Upon reaching the end of the project, it was observed that the neural network had a certain failure rate in predicting moments of heavy rainfall due to the lack of data of this type for its training. It is also considered that if it is possible to obtain more data from IPMA, whether provided by the organization itself or obtained by the developer during periods of frequent and intense rainfall, the algorithm would achieve a good accuracy rate.

Keywords: Deep Learning, CNN, Meteorology, IPMA, Artificial Neural Networks

Índice

Lista de Figuras

Lista de tabelas

Lista de siglas e acrónimos

API	*Application Programming Interface*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
CSS	*Cascading Style Sheets*
DOM	*Document Object Model*
GPU	*Graphic Processing Unit*
GUI	*Graphical User Interface*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
IPMA	Instituto Português do Mar e da Atmosfera
JSON	*JavaScript Object Notation*
KB	*kilobyte*
ML	*Machine Learning*
NLP	*Natural Language Processing*
ReLU	*Rectified Linear Unit*
RGBA	*Red-Green-Blue-Alpha*
TPU	*Tensor Processing Unit*
URL	*Uniform Resource Locator*
1D	*Uma dimensão*

1. Introdução

A previsão do tempo é uma área multidisciplinar que combina observações atmosféricas, técnicas de modelagem e processamento de dados para estimar as condições climáticas futuras. Ela envolve a obtenção de informações de várias fontes, como satélites, radares, estações terrestres, entre outros, para entender o estado atual da atmosfera e fazer projeções sobre como ela poderá evoluir nas próximas horas, dias ou até mesmo semanas.

Os dados recolhidos incluem informações como intensidade do vento, temperatura, humidade, pressão atmosférica, vento, precipitação e padrões climatéricos. Essas informações são alimentadas em modelos computacionais que utilizam equações matemáticas complexas para simular o comportamento da atmosfera. Com base nessas simulações, as previsões do tempo são geradas e refinadas à medida que mais dados são incorporados ao longo do tempo.

No contexto deste projeto, o foco está na previsão de curto prazo, com base nos dados dos radares meteorológicos do IPMA e técnicas avançadas de processamento de imagens. As redes neuronais convolucionais são um tipo de algoritmo de *machine learning* (ML) que se mostram eficazes no reconhecimento de padrões de imagens. No caso específico, a *Convolutional Neural Network* (CNN) é treinada para identificar o nível de precipitação numa escala de milímetros por hora com base nas imagens de radar.

A abordagem de usar tecnologias de ML e processamento de imagens visa melhorar a precisão das previsões meteorológicas. Com previsões mais confiáveis e detalhadas sobre a precipitação, é possível tomar decisões mais informadas em diversas áreas. Por exemplo, na agricultura, os agricultores podem ajustar os seus planos de irrigação, e relativamente aos meios de transporte, as pessoas podem planear melhor as suas rotas, escolher o meio de transporte adequado e preparar-se para possíveis condições adversas.

Além disso, as previsões meteorológicas precisas também têm um impacto direto na segurança, permitindo que as autoridades emitam alertas antecipados de tempestades, inundações ou outros eventos climatéricos extremos. Isso dá às comunidades tempo suficiente para se prepararem e tomarem as devidas medidas preventivas, reduzindo assim os danos e riscos associados a esses eventos.

No geral, o uso destas tecnologias avançadas, combinadas com dados meteorológicos precisos, tem o potencial de fornecer previsões meteorológicas mais confiáveis e detalhadas, beneficiando tanto a sociedade em geral, quanto diversos setores económicos, que dependem dessas informações para tomar decisões estratégicas.

2. Enquadramento teórico

2.1. *Deep Learning*

Deep Learning é uma técnica poderosa no campo da inteligência artificial que visa replicar o funcionamento do cérebro humano. Para explicar o *Deep Learning* de forma simples, envolve a construção de redes neuronais com três ou mais camadas (Figura 1). Essas redes neuronais são projetadas para analisar grandes quantidades de dados e "aprender" com eles.

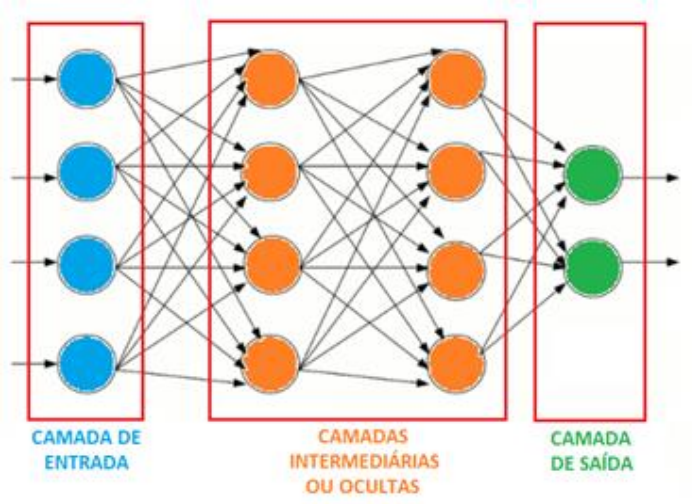

Figura 1 – ***Rede neuronal com quatro camadas.***

O objetivo principal do *Deep Learning* é permitir que as máquinas façam previsões precisas e realizem tarefas sem a intervenção humana. Ao imitar o comportamento do cérebro humano, os algoritmos de *Deep Learning* podem processar e interpretar padrões e características complexas nos dados, permitindo-lhes identificar e classificar informações com uma precisão cada vez maior.

A implementação da tecnologia de *Deep Learning* tem levado a avanços significativos em várias aplicações e serviços. Produtos e serviços do dia a dia, como assistentes virtuais, comandos de TV por voz e sistemas de deteção de fraude em cartões de crédito, dependem de algoritmos de *Deep Learning*. Além disso, tecnologias emergentes, como carros automáticos, utilizam amplamente o *Deep Learning* para a sua operação.

Ao aproveitar o potencial do *Deep Learning*, as máquinas tornam-se capazes de realizar tarefas analíticas que anteriormente exigiam intervenção humana. Esta tecnologia tem revolucionado a automação, abrindo o caminho para sistemas mais inteligentes e eficientes em diferentes setores [1].

2.1.1. O neurónio

Os neurónios artificiais são inspirados pela biologia e tentam replicar o funcionamento do nosso cérebro. O nosso cérebro possui mais de cem mil milhões de células neuronais que nos ajudam a interpretar os chamados sinais ou sinapses.

Pode-se imaginar uma sinapse como qualquer coisa que varia desde um raio de luz que atinge os nossos olhos, até ao simples pensamento de que queremos mover o braço. Os neurónios ajudam-nos a interpretar esses sinais de forma correta.

Agora, neurónios artificiais são uma abordagem para transferir esse princípio para um computador, replicando um neurónio biológico em forma de código. (Figura 2)

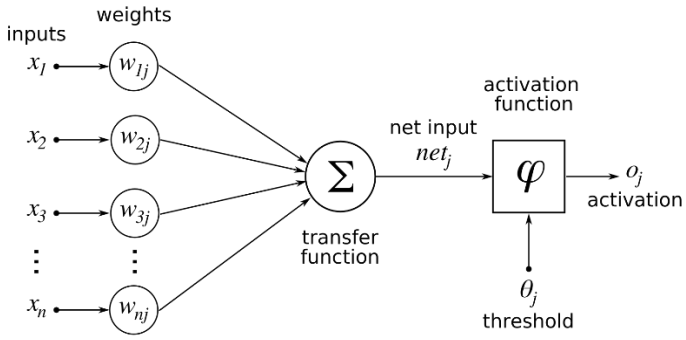

Figura 2 - O neurónio artificial.

Para replicar este processo surgiu um novo conceito, os pesos. Um neurónio artificial começa por receber um input, de seguida atribui um peso a esse *input* consoante a sua importância e multiplica o *input* pelo peso gerado, criando assim um novo valor.

No contexto de um neurónio artificial, o *bias* refere-se a um parâmetro adicional que é incorporado no cálculo da saída do neurónio. Pode ser considerado como um valor constante que é adicionado à soma ponderada das entradas antes de passar por uma função de ativação.

O termo *bias* permite ao neurónio artificial ter influência na sua saída, mesmo quando todos os valores de entrada são zero. Ele fornece ao neurónio a capacidade de deslocar o limiar da função de ativação e afetar a resposta geral do neurónio. Ajustando o *bias*, o neurónio pode se tornar mais ou menos propenso a ativar, mesmo na ausência de sinais de entrada fortes. O peso do *bias* é ajustado durante o processo de treino juntamente com os outros pesos das conexões para otimizar o desempenho do neurónio artificial [2].

De seguida, a soma do peso multiplicado pelo *input* e do *bias* passa por uma função de ativação.

Uma função de ativação de uma rede neuronal é responsável por determinar a saída ou ativação de um neurónio artificial com base na entrada recebida. Esta define se o neurónio deve ser ativado ou não, influenciando se a informação será transmitida para os neurónios subsequentes na rede.

Em geral, uma função de ativação compara a entrada recebida com um limite ou *threshold*. Se a entrada exceder esse limite, o neurónio é ativado e a sua saída é propagada para os neurónios seguintes. Caso contrário, se a entrada for menor ou igual ao limite, o neurônio permanece inativo e não transmite informações.

O *threshold* determina o nível mínimo necessário para que o neurónio dispare, ou seja, para que ele seja ativado. É uma medida de sensibilidade do neurônio em relação aos estímulos que recebe. O *threshold* pode variar dependendo da função de ativação escolhida e das necessidades do problema em questão.

A função de ativação pode assumir diferentes formas e características e cada uma delas possui propriedades distintas que influenciam o comportamento do neurónio e a capacidade da rede neuronal de aprender e representar relações complexas.

Portanto, a função de ativação desempenha um papel crucial na tomada de decisão dos neurónios artificiais, permitindo que a rede neuronal processe informações, aprenda padrões e faça previsões com base nos dados de entrada [3].

2.1.2. A rede neuronal

Uma rede neuronal é composta por um conjunto de camadas interligadas. Cada camada é formada por neurónios que recebem parâmetros de entrada e passam por um processo de atribuição de pesos e uma aplicação de uma função de ativação [4].

O processo começa na camada de entrada, onde os neurónios recebem os valores dos parâmetros de entrada. De seguida, cada neurónio dessa camada passa o seu valor para todos os neurónios da próxima camada, acontecendo dessa forma a chamada *forward propagation* [5].

À medida que os valores dos neurónios são propagados pela rede, eles passam pelo processo de atribuição de pesos e função de ativação em cada camada subsequente. Os pesos determinam a importância relativa dos valores de entrada para cada neurónio, enquanto a função de ativação determina o valor de saída do neurónio com base nos valores ponderados de entrada.

Esse processo continua até que os valores atinjam a camada final da rede. Nessa camada, os neurónios aplicam o mesmo processo descrito anteriormente, atribuindo pesos aos valores de entrada e aplicando uma função de ativação. Os resultados obtidos são utilizados como as saídas da rede neuronal.

Dependendo da tarefa em questão, pode-se utilizar uma função de ativação específica na camada de saída. Por exemplo, em problemas de classificação binária, é comum utilizar a função *sigmoid*, que produz uma saída entre 0 e 1, representando a probabilidade de pertencer a uma classe específica. Outra função de ativação usada é a *softmax*, na qual o *output* de cada neurónio da camada final representa a probabilidade de o *input* pertencer à classe associada a esse neurónio [6].

Resumindo, uma rede neuronal é uma estrutura composta por camadas interligadas de neurónios. Cada neurónio realiza o processo de atribuição de pesos e função de ativação, passando os valores para os neurónios da próxima camada. Esse processo repete-se até à camada final, onde são geradas as saídas da rede neuronal. A escolha da função de ativação adequada para a tarefa em questão é fundamental para obter resultados precisos e adequados ao problema.

2.1.3. Aplicações

As aplicações em *Deep Learning* estão presentes no nosso quotidiano, embora, na maioria dos casos, estão tão bem integradas nos produtos e serviços que os utilizadores não têm conhecimento do complexo processamento de dados que ocorre.

Alguns exemplos destas aplicações são:

- Aplicação na área de segurança pública:

Os algoritmos *Deep Learning* podem analisar e aprender com dados transacionais para identificar padrões perigosos que indiquem possíveis atividades fraudulentas ou criminosas. Aplicações de reconhecimento de voz, visão computacional e outras áreas do *Deep Learning* podem melhorar a eficiência e eficácia da análise investigativa, extraindo padrões e evidências de gravações de som e vídeo, imagens e documentos. Isto ajuda as forças de segurança a analisar grandes quantidades de dados de forma mais rápida e precisa.

- Aplicação na área de serviços financeiros:

As instituições financeiras utilizam regularmente análises preditivas para impulsionar a negociação algorítmica de ações, avaliar riscos empresariais para a aprovação de empréstimos, detetar fraudes e auxiliar na gestão do crédito e carteiras de investimento para os clientes.

- Aplicação na área de atendimento ao cliente:

Muitas organizações incorporam tecnologia com *Deep Learning* nos seus processos de atendimento ao cliente. Os *chatbots*, amplamente utilizados em várias aplicações, serviços e portais de atendimento ao cliente, são uma forma de inteligência artificial. Os *chatbots* tradicionais usam linguagem humana e, até mesmo, reconhecimento visual, geralmente encontrados em centrais de atendimento.

No entanto, soluções de *chatbot* mais sofisticadas tentam determinar, se existem múltiplas respostas para perguntas indeterminadas. Com base nas respostas recebidas, o *chatbot* tenta responder a essas perguntas diretamente ou encaminhar a conversa para um utilizador.

Assistentes virtuais, como a *Siri* da *Apple*, *Alexa* da *Amazon* ou o *ChatGPT* da *OpenAI*, ampliam a ideia de um *chatbot*, permitindo a funcionalidade de reconhecimento de voz. Isto cria um novo método para envolver utilizadores de forma personalizada.

- Aplicação na área da saúde:

A indústria da saúde tem se beneficiado imensamente das capacidades do *Deep Learning* desde a digitalização dos registos e imagens hospitalares. Aplicações de reconhecimento de imagens podem apoiar especialistas em imagiologia médica e radiologistas, ajudando-os a analisar e avaliar mais imagens em menos tempo.

Estas são apenas algumas das aplicações reais que usam *Deep Learning* que estão a transformar diferentes setores da sociedade, melhorando a eficiência, precisão e qualidade dos serviços prestados. Com o avanço contínuo da tecnologia, espera-se que o campo do *Deep Learning* continue a evoluir e a encontrar novas formas de melhorar a nossa vida diária [7].

2.2.*Convolutional Neural Network*

Uma *Convolutional Neural Network*, também conhecida como CNN ou *ConvNet*, é uma classe de redes neuronais que se especializa no processamento de dados e que possuem uma topologia em forma de grade, como uma imagem [8].

Uma imagem digital é uma representação binária de dados visuais. Ela contém uma série de *pixels* dispostos numa estrutura semelhante a uma matriz, onde os valores dos *pixels* indicam o brilho e a cor de cada *pixel* (Figura 3).

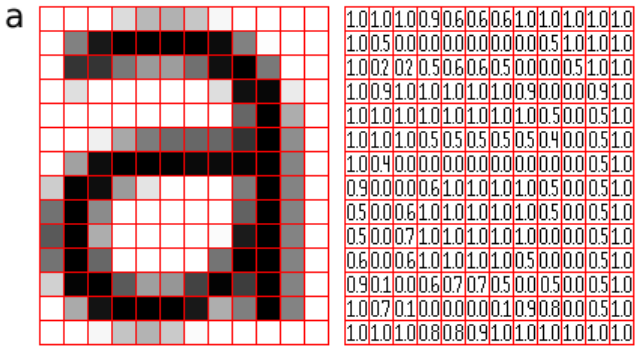

Figura 3 - Representação de uma imagem como matriz de *pixels.*

As CNN têm vários tipos de camadas específicas que organizadas de uma certa forma têm a capacidade de reconhecer padrões e características simples, como bordas e texturas e até mesmo coisas mais complexas como objetos ou pessoas. De seguida será explicado as camadas mais importantes deste tipo de rede.

2.2.1. Convolution Layer

A camada convolucional, ou *Conv2D*, é a base de todas as CNN's. É boa prática começar uma CNN com uma camada deste tipo, pois reduz os dados significativa e eficientemente, passando para as camadas que a sucedem os dados resumidos, mas pertinentes, diminuindo assim o custo computacional da rede, mas não afetando a sua performance.

O parâmetro mais importante duma camada convolucional é o seu filtro, também conhecido como *kernel*. Este filtro tem o seu tamanho predefinido, sendo normalmente 2x2, 3x3 ou 5x5 e é o responsável por gerar o mapa de características, sendo esse o output desta camada.

Após receber uma matriz de *pixels*, o filtro vai deslizando ao longo da matriz. Para cada posição do *kernel*, os valores dos pixels cobertos pelo mesmo, são multiplicados pelos seus próprios valores. O resultado dessas multiplicações é somado dando assim origem ao novo valor do mapa de características. Ao fazer isto o mapa de características é capaz de detetar os pontos mais relevantes de uma certa imagem (Figura 4).

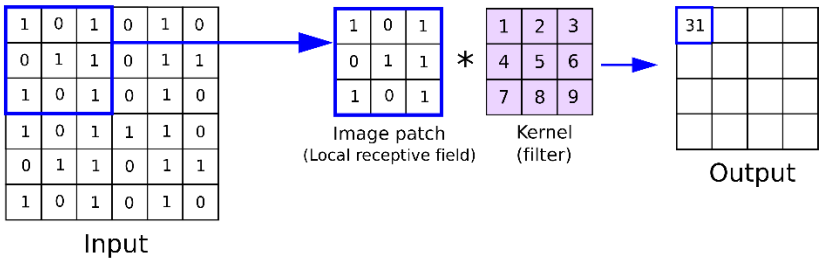

Figura 4 - *Convolutional Layer.*

O mapa de caraterísticas pode ainda ser subjugado a uma função de ativação para extrair ainda mais os pontos mais importantes de uma certa imagem [9].

2.2.2. Pooling Layer

A *Pooling layer*, normalmente subsequente à *Convolutional Layer*, tem como objetivo reduzir o tamanho do mapa de características, gerando como *output* uma versão reduzida do que recebeu.

Tal como a *Convolutional Layer*, a *Pooling Layer* também recebe um filtro, mas desta vez conhecido como *Pool size*, que passará pelo mapa de caraterísticas. Existindo dois tipos de camadas de *Pooling*, o filtro tem um funcionamento diferente em cada uma delas.

No caso da *Max Pooling Layer*, o filtro seleciona o valor mais alto de dentro dos que estão a ser cobertos por ele e passa para o novo mapa de características.

Já na *Average Pooling Layer*, o filtro faz a média dos valores que estão a ser cobertos por ele, sendo o resultado, o novo valor no mapa de características (Figura 5).

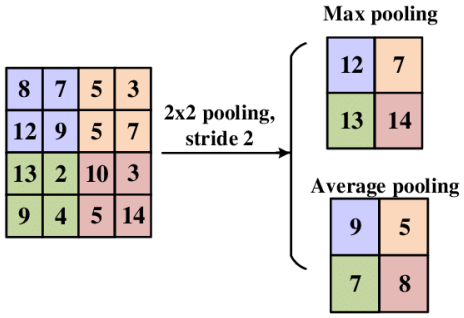

Figura 5 - *Pooling Layer types.*

Esta camada é utilizada especialmente para reduzir o número de parâmetros da rede de forma a otimizar ainda mais o modelo. No entanto também têm alguns efeitos adversos como a perda de informação detalhada [10].

2.2.3. Fully connected layer

A *Fully connected layer* não tem nenhum requisito para o parâmetro de entrada e usualmente é utilizada como a última camada de uma CNN. A estrutura da *Fully connected layer* pode variar dependendo do objetivo da rede.

No caso de classificação binária, onde a rede deve determinar se uma imagem pertence a uma determinada classe ou não, a camada consistirá em apenas um neurónio com a função de ativação *sigmoid*. Esse neurônio calculará um valor entre 0 e 1, representando a probabilidade de a imagem pertencer à classe em questão. Um valor próximo de 1 indica que a rede está confiante de que a imagem pertence à classe, enquanto um valor próximo de 0 indica que a rede acredita que a imagem não pertence à classe.

Por outro lado, no caso da classificação para várias classes, onde a rede deve atribuir uma imagem a uma de várias classes possíveis, a camada terá tantos neurónios quanto o número de classes existentes. Cada neurónio estará associado a uma classe específica e calculará a probabilidade de a imagem pertencer a essa classe. Para obter uma distribuição de probabilidade correta, é comum aplicar a função de ativação *softmax* nos valores de saída desses neurónios. A função *softmax* normaliza os valores de saída para que eles somem 1, fornecendo assim uma probabilidade para cada classe [11].

3. Descrição do processo

3.1. Ferramentas utilizadas

Para o desenvolvimento deste projeto de grande magnitude, foram selecionadas ferramentas essenciais que desempenharão um papel fundamental em todo o processo. O uso do *GitHub Desktop*, do *Jupyter Notebook* e do *Visual Studio Code* foi cuidadosamente decidido, levando em consideração a sua eficiência, facilidade de uso e recursos avançados.

3.1.1. *GitHub Desktop*

O *GitHub Desktop* é uma *graphical user interface* (GUI) desenvolvida pelo *GitHub* para facilitar o uso do *Git*, um sistema de controlo de versões amplamente utilizado para gerir o código-fonte de projetos de *software*. O *GitHub Desktop* fornece uma maneira fácil e visualmente intuitiva (Figura 6) de trabalhar com repositórios *Git* no *GitHub*.

Com o *GitHub*, a equipa terá a capacidade de seguir as alterações feitas no projeto, analisar e discutir propostas, e fornecer *feedback* de forma eficiente. Além disso, o *GitHub* oferece recursos para gestão de problemas e tarefas, permitindo que a equipa mantenha o controlo do progresso do projeto (Figura 6) [12].

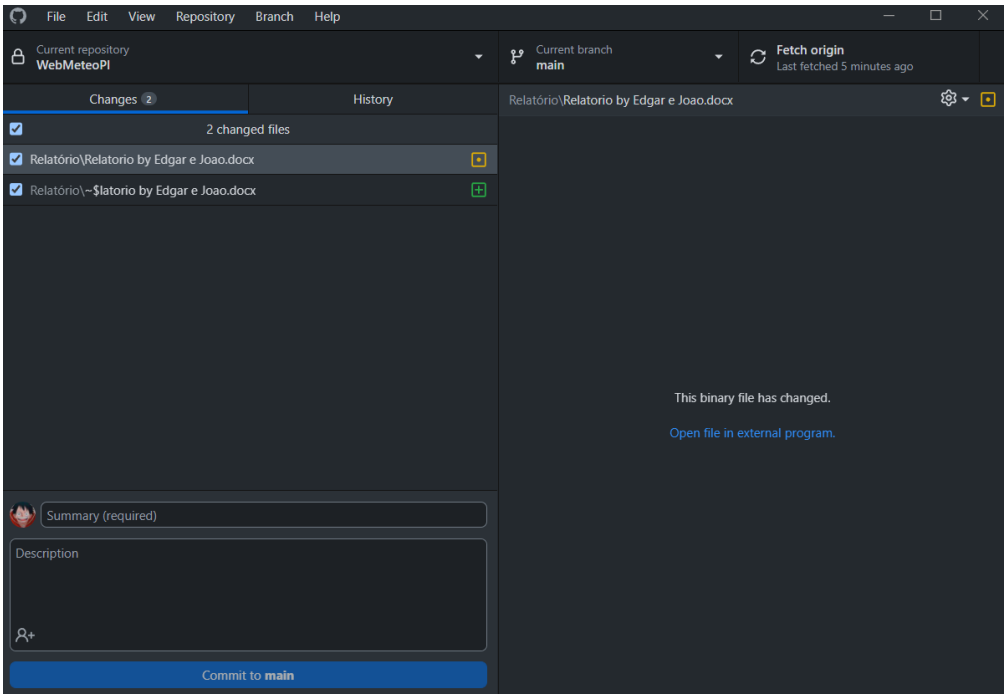

Figura 6 - *GitHub Desktop.*

3.1.2. *Jupyter Notebook*

O *Jupyter Notebook* é uma *open-source web application* que permite criar e compartilhar documentos interativos que contêm código, visualizações e texto explicativo. Este é amplamente utilizado na ciência de dados, pesquisa científica e educação, pois oferece uma maneira conveniente de combinar código executável com elementos narrativos (Figura 7).

A principal característica do *Jupyter Notebook* é o suporte a células, que são unidades independentes onde é possível escrever e executar código, bem como adicionar texto, imagens e gráficos. Isso torna o processo de desenvolvimento iterativo e exploratório, permitindo que seja executado e visualizado os resultados do código em tempo real.

As células podem conter código em várias linguagens de programação e quando são executadas, o código dentro dela é processado e o resultado é exibido imediatamente abaixo da célula. Isso facilita ao testar o código, pois é possível ver os resultados intermédios à medida que avança [13].

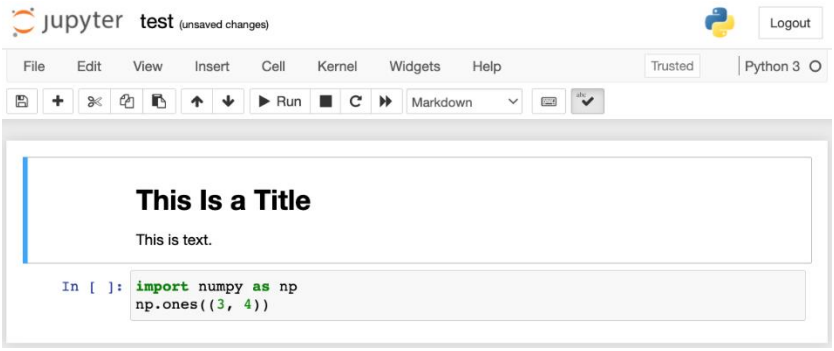

Figura 7 - *Jupyter Notebook*.

3.1.3. Visual Studio Code

O *Visual Studio Code* é um *Integrated Development Environment* (IDE) criado pela *Microsoft*, que ganhou uma enorme popularidade entre os programadores. Com a sua *user interface* amigável e extensibilidade, o *Visual Studio Code* oferece suporte a uma ampla variedade de linguagens de programação. Este possui recursos essenciais, como realce de sintaxe, formatação de código e *debug*, para facilitar o desenvolvimento de aplicações de qualquer escala. Além disso, o *Visual Studio Code* é altamente personalizável por meio de extensões, o que permite aos utilizadores adicionar funcionalidades extras, integração com ferramentas externas e suporte a *frameworks* específicas.

Uma característica notável deste IDE é o seu terminal integrado, que elimina a necessidade de alternar entre o editor e uma janela do terminal separada. Além disso, o IDE oferece recursos avançados de controlo de versões, sendo até possível integrar o *GitHub* dentro do mesmo, permitindo aos programadores gerir repositórios *Git* diretamente dentro do ambiente de desenvolvimento [14].

3.1.4. *Flask*

O *Flask* é uma *framework web* leve e flexível desenvolvido em *Python*. Ele também é conhecido como uma *micro-framework*, o que significa que fornece apenas o básico necessário para criar aplicações web, tornando-o fácil de aprender e usar.

O *Flask* possui uma *Application Programming Interface* (API) minimalista e simples, permitindo que seja possível criar aplicações web com menos código e menos complexidade. Ela oferece um *routing system* que mapeia *Uniform Resource Locators* (URLs) para funções específicas, determinando como a aplicação deve responder aos seus pedidos. Também é uma *framework* altamente extensível, o que significa que é possível adicionar funcionalidades extras através de extensões ou bibliotecas de terceiros.

Além disso, o *Flask* possui uma documentação abrangente e bem escrita, que fornece exemplos claros e tutoriais passo a passo para ajudar os desenvolvedores a começar e aprofundar seus conhecimentos.

Resumindo, o *Flask* é um *framework* web leve, flexível e fácil de usar, que permite que sejam criadas aplicações *web* usando a linguagem *Python*. Este é uma ótima opção para projetos pequenos a médios, onde a simplicidade e a facilidade na aprendizagem são prioridades [15].

3.1.5. Vue.js

O *Vue.js* é uma *framework* JavaScript que permite criar interfaces de utilizador de forma simples, eficiente e versátil. Baseia-se em *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS) e *JavaScript* padrão e oferece um modelo de programação declarativo e baseado em componentes que facilita o desenvolvimento de interfaces, sejam elas simples ou complexas. Além disso, o *Vue.js* é progressivo, o que significa que pode ser adotado de forma incremental, integrando-se com outras bibliotecas ou *frameworks* conforme a necessidade.

Algumas das vantagens sobre o *Vue.js* são:

1. **Simplicidade**

O *Vue.js* é fácil de aprender e usar, pois requer apenas conhecimentos básicos de HTML, CSS e *JavaScript*. Não é necessário aprender outras linguagens ou ferramentas específicas, como acontece com outras *frameworks* como o *Angular* e o *React*. Além disso, o *Vue.js* tem uma sintaxe clara e intuitiva, que permite escrever aplicações com poucas linhas de código.

2. **Arquitetura baseada em componentes**

O *Vue.js* segue uma arquitetura baseada em componentes, que consiste em dividir o código da aplicação em partes independentes e reutilizáveis. Cada componente contém um *template*, a lógica e os estilos relacionados à sua funcionalidade. Os componentes podem ser compostos entre si para formar a aplicação completa. Essa abordagem facilita a organização, a manutenção e o teste do código.

3. **Reatividade**

O *Vue.js* possui um sistema de reatividade que permite atualizar automaticamente a interface de utilizador quando o estado da aplicação muda. Isto é feito através de um mecanismo de observação e notificação que deteta as mudanças nos dados e as reflete no *Document Object Model* (DOM). Assim, não é necessário manipular manualmente o DOM ou usar bibliotecas externas para isso.

4. **Desempenho**

O *Vue.js* é uma *framework* leve e rápida, que ocupa cerca de 20 *kilobytes* (KB) comprimido. Usa um DOM virtual para renderizar os componentes, o que reduz o número de operações no DOM real e melhora o desempenho da aplicação. Além disso, o *Vue.js* utiliza técnicas de compilação e otimização que permitem gerar código eficiente e adaptado ao navegador.

5. **Ecossistema**

O *Vue.js* conta com um ecossistema rico e diverso, que oferece soluções para diversas funcionalidades avançadas, como *routing*, gestão de estado, validação de formulários, renderização no lado do servidor, geração de *websites* estáticos, entre outros. Essas soluções são mantidas pela equipa oficial do *Vue.js* ou pela comunidade de programadores, que contribui com projetos *open source* e recursos educacionais [16].

3.1.6. *Tensorflow*

Amplamente utilizado por *data scientists* e *software developers*, o TensorFlow é uma plataforma de código aberto para ML usando grafos de fluxo de dados. Os nós no grafo representam operações matemáticas, enquanto as arestas do grafo representam as matrizes de dados multidimensionais (*tensors*) que fluem entre eles. Esta arquitetura flexível permite que algoritmos de ML sejam descritos como um grafo de operações conectadas.

Estes podem ser treinados e executados em Graphics Processing Units (GPUs), *Central Processing Units* (CPUs) e *Tensor Processing Units* (TPUs) em várias plataformas sem precisar reescrever o código, abrangendo desde dispositivos portáteis até *desktops* e servidores de alta qualidade. Isto significa que programadores de todos os níveis podem usar as mesmas ferramentas para colaborar, aumentando significativamente a sua eficiência.

Desenvolvido inicialmente pela equipa da *Google Brain* para fins de pesquisa em ML e redes neuronais, o sistema também é versátil o suficiente para ser aplicável em uma ampla variedade de outros domínios.

O *TensorFlow* pode ser usado para desenvolver modelos para várias tarefas, incluindo *Natural Language Processing* (NLP), reconhecimento de imagem, reconhecimento de escrita à mão e diferentes simulações baseadas na computação, como equações diferenciais parciais.

Os principais benefícios do *TensorFlow* estão na sua capacidade de executar operações de baixo nível em várias plataformas de aceleração, cálculo automático de gradientes, escalabilidade em nível de produção e exportação de gráficos interoperáveis. Ao fornecer o *Keras* como uma API de alto nível e a execução ágil como uma alternativa ao paradigma de fluxo de dados no *TensorFlow*, é sempre fácil escrever código confortavelmente.

Como criador original do *TensorFlow*, a *Google* ainda apoia fortemente a biblioteca e tem impulsionado o ritmo acelerado no seu desenvolvimento. Por exemplo, o *Google* criou um *hub online* para compartilhar os muitos modelos diferentes criados pelos utilizadores [17].

3.2. *Dataset* utilizado

Para o desenvolvimento deste projeto, foi necessário criar um conjunto de dados próprio, devido às restrições de disponibilidade de dados.

As imagens utilizadas foram obtidas (Figura 8) a partir dos radares de precipitação do IPMA, sendo disponibilizadas apenas imagens com duração máxima de cerca de um mês.

```
url_image = f"https://www.ipma.pt/resources.www/transf/radar/por/pcr-{date_time_utc.strftime('%Y-%m-%d')}T{date_time_utc.strftime('%H%M')}.png"
response = requests.get(url_image)
image_data = BytesIO(response.content)
image = Image.open(image_data)
```

Figura 8 - Código para obter imagem para o *dataset.*

Já os dados sobre os valores de precipitação das respetivas estações que o IPMA dispõe, são fornecidos os das 3 horas antecedentes à hora certa mais recente. Por exemplo, se for feito um pedido à API às 14:00h, serão obtidos os dados das 11:00h, 12:00h e 13:00h de todas as estações.

Ainda assim, dado que ao longo do desenvolvimento do projeto existiram algumas alterações no formato *standard* das imagens fornecidas pelo IPMA, sendo que, inicialmente as imagens vinham com as fronteiras delineadas a preto, mas, após um certo período, essas fronteiras foram removidas, e, eventualmente, voltaram a ser colocadas, nós desenvolvemos uma função capaz de remover os *pixels* pretos de uma imagem fazendo com que o *dataset* construído seja mais uniforme (Figura 9).

```
def remove_black_pixels(image):
    # Convert the image to RGBA mode (if it's not already in RGBA mode)
    image = image.convert("RGBA")
    # Get the pixel data as a list of tuples
    pixels = list(image.getdata())
    # Replace every black pixel with transparent
    new_pixels = []
    for pixel in pixels:
        if pixel[0] == 0 and pixel[1] == 0 and pixel[2] == 0:
            new_pixels.append((0, 0, 0, 0))
        else:
            new_pixels.append(pixel)
    # Create a new image with the same size and mode as the original image
    new_image = Image.new(image.mode, image.size)
    # Update the new image with the new pixel data
    new_image.putdata(new_pixels)
    # Return the new image
    return new_image
```

Figura 9 - Função para remover *pixels* pretos das imagens.

Dado que as imagens fornecidas pelo IPMA contêm o mapa de Portugal Continental na sua integra e a API fornece dados relativos a várias estações, para obter mais dados para treino, foi feito o recorte do mapa para cada distrito atribuindo a cada imagem o valor de precipitação da estação mais próxima da capital de distrito.

Foi feito um recorte de duzentos por duzentos *pixels* de forma a cobrir o distrito na sua totalidade. Para chegar a esta finalidade foram realizados dezenas de testes de forma a ser encontrada a posição ideal para o recorte de cada um dos dezoito distritos de Portugal Continental de forma a que cobrisse o distrito na sua integra.

Apesar disto, apenas era possível obter um máximo de 432 elementos para o *dataset* diariamente, sendo a maioria esmagadora pertencente à mesma classe, então, por parte dos orientadores, foi recomendado serem realizadas técnicas de *data augmentation*.

3.2.1. *Data augmentation*

Data augmentation é uma técnica utilizada para aumentar artificialmente o *dataset* para o treino, criando cópias modificadas do *dataset* original.

É uma técnica bastante utilizada em problemas de *deep learning*, por ser a única solução para um dos maiores problemas que as redes neuronais ainda apresentam, que é a dependência sob dados de treino para atingir o seu bom funcionamento.

Para *datasets* baseados em imagens, algumas das técnicas mais famosas são realizar transformações geométricas nas imagens, alterar as cores das mesmas, apagar partes aleatórias, entre outras, mas sendo que estes dados são mais 'delicados' e não seria lógico dentro do contexto do problema fazer grandes alterações às imagens teve-se uma abordagem mais suave [16].

Essas alterações foram as seguintes:

1. Deslocamento da imagem em 20 *pixels* para baixo, copiando a "linha" superior da imagem para as 20 novas linhas criadas (Figura 10);

```python
def shift_image_down(image_path, shift_value):
    # Open the image
    image = Image.open(image_path)

    # Create a new image with the updated height
    new_image = Image.new(image.mode, (image.width, image.height))

    # Copy the original image onto the new image, shifted down
    new_image.paste(image, (0, shift_value))

    # Get the first row of the original image
    first_row = image.crop((0, 0, image.width, 1))

    # Paste the first row into the new rows
    for y in range(shift_value):
        new_image.paste(first_row, (0, y, image.width, y + 1))

    # Return the shifted and filled image
    return new_image
```

Figura 10 - Código da função para deslocar a imagem para baixo.

2. Deslocamento da imagem em 20 *pixels* para a direita, copiando a "linha" da esquerda da imagem para as 20 novas colunas criadas (Figura 11);

```python
def shift_image_left(image_path, shift_value):
    # Open the image
    image = Image.open(image_path)

    # Create a new image with the updated width
    new_image = Image.new(image.mode, (image.width, image.height))

    # Copy the original image onto the new image, shifted to the left
    new_image.paste(image, (-shift_value, 0))

    # Get the last column of the original image
    last_column = image.crop((image.width - 1, 0, image.width, image.height))

    # Paste the last column into the new columns
    for x in range(image.width - shift_value, image.width):
        new_image.paste(last_column, (x, 0, x + 1, image.height))

    image.close()
    # Return the shifted and filled image
    return new_image
```

Figura 11 - Código da função para deslocar a imagem para a esquerda.

3. Deslocamento da imagem em 20 *pixels* para a esquerda, copiando a "linha" da direita da imagem para as 20 novas colunas criadas (Figura 12);

```python
def shift_image_right(image_path, shift_value):
    # Open the image
    image = Image.open(image_path)

    # Create a new image with the updated width
    new_image = Image.new(image.mode, (image.width, image.height))

    # Copy the original image onto the new image, shifted to the right
    new_image.paste(image, (shift_value, 0))

    # Get the first column of the original image
    first_column = image.crop((0, 0, 1, image.height))

    # Paste the first column into the new columns
    for x in range(shift_value):
        new_image.paste(first_column, (x, 0, x + 1, image.height))

    image.close()
    # Return the shifted and filled image
    return new_image
```

Figura 12 - Código da função para deslocar a imagem para a direita.

4. Rotação da imagem original em 1 grau (Figura 13);

```python
def rotate_image(image_path, angle):
    # Open the image
    image = Image.open(image_path)

    # Rotate the image
    rotated_image = image.rotate(angle, expand=True)

    # Create a new image with RGBA mode and transparent background
    new_image = Image.new("RGBA", rotated_image.size, (0, 0, 0, 0))

    # Paste the rotated image onto the new image
    new_image.paste(rotated_image, (0, 0), rotated_image)

    #Resize image to 200x200
    new_image = new_image.resize((200,200))

    image.close()
    # Return the rotated and filled image
    return new_image
```

Figura 13 - Código da função para rodar a imagem.

5. Rotação da imagem gerada pelo primeiro método (deslocamento para baixo) em 1 grau;
6. Rotação da imagem gerada pelo segundo método (deslocamento para a direita) em 1 grau;
7. Rotação da imagem gerada pelo terceiro método (deslocamento para a esquerda) em 1 grau;

Estas técnicas de aumento de dados permitem criar variações das imagens originais, ampliando o conjunto de dados disponível para o treino do modelo. Com uma maior diversidade de imagens, espera-se que o modelo seja capaz de aprender padrões mais abrangentes e generalizáveis relacionados à precipitação.

Apesar de todos os métodos empregues, não foi possível obter imagens de chuva intensa devido ao estado meteorológico predominante em Portugal Continental ter sido, maioritariamente, ameno.

3.2.2. Normalização dos dados

Já para os valores de precipitação, com o objetivo de obter um treino mais estável e preciso, optou-se por normalizar o valor da precipitação com base no maior valor já registado em Portugal pelo IPMA, que foi de 220 milímetros cúbicos por hora, em Penhas da Saúde no dia catorze de janeiro de mil novecentos e setenta e sete. Ainda assim, decidiu-se aumentar esse valor para proporcionar uma margem de segurança para possíveis valores extremos que possam ocorrer [17].

A fórmula utilizada para essa normalização foi a seguinte:

$$Valor\ normalizado = \frac{Valor\ original}{240 * 100}$$

Como medida preventiva, optou-se por arredondar o valor obtido. Além disso, caso o valor fosse negativo, substituía-se por zero. Essa medida foi implementada devido a alguns pedidos que continham o valor de "-99.0", por apresentar erros provenientes da API do IPMA (Figura 14).

```
def normalize_precipitation_value(precipitation_value):
    return int(round((precipitation_value/240)*100,0))
```

Figura 14 - Código da função de normalização do valor de precipitação.

Sendo assim, devido à escassez de dados, após a normalização dos dados, foram apenas obtidos dados com os valores de 0, 1, 2, 3 e 5 como label. Os valores finais de precipitação (Tabela 1) obtidos, ou seja, o número de imagens com labels sendo a mesma o seu valor de precipitação normalizado, foram os seguintes:

Tabela 1 - Total de dados obtidos

Quantidade de imagens em que o label é 0	Quantidade de imagens em que o label é 1	Quantidade de imagens em que o label é 2	Quantidade de imagens em que o label é 3	Quantidade de imagens em que o label é 5
21592	408	168	16	8

Estas imagens e valores foram utilizados tanto para o treino quanto para os testes do modelo construído (Anexo 2 - Código da função de obtenção de imagens e valores do IPMA e variáveis uteis para o bom funcionamento da mesma).

3.3. Construção da rede neuronal

3.3.1. Arquitetura da rede neuronal

A versão final do modelo é constituída por um total de treze camadas. Primeiramente, para receber o *input*, existe uma camada Convolucional 2D com dezasseis neurónios, um *max pooling size* de (2,2), a *Rectified Linear Unit* (ReLU) como função de ativação e, por fim, um *input shape* de 100×100 *pixels* com quatros canais, dado que, as imagens fornecidas pelo IPMA, vêm no formato *Red-Green-Blue-Alpha* (RGBA). De seguida, outra camada Convolucional 2D, também com dezasseis neurónios, um *filter size* de 3, o *stride* com tamanho de 1, o *padding* com o valor '*same*' e também a ReLU como função de ativação. Foi decidido começar o modelo com duas camadas Convolucional 2D, pois, ao seguir essa mesma abordagem, o modelo tem capacidade de detetar características mais complexas e abstratas. A primeira camada é responsável por capturar características mais simples como bordas, formas básicas, entre outras, enquanto a segunda camada tem como intuito capturar características mais complexas que se integram nas características da camada precedente, revelando assim, os padrões mais significativos das imagens.

Após estas duas camadas está uma cada de *BatchNormalization*. Este tipo de camada tem como objetivo melhorar a estabilidade e velocidade do treino. Esta camada é responsável por normalizar o *input*, ajustando o mesmo a uma certa escala.

Após isto, foi posta uma cada de *Dropout* com o valor de 0.6. Esta camada, é responsável por substituir 60% dos valores de *input* por 0. A mesma foi adicionada por ser a camada indicada para evitar o *overfitting*, um problema muito comum em redes neuronais.

De seguida, está uma camada de *MaxPooling* com o *pool size* de (2,2), e outra camada de *BatchNormalization*. A camada *MaxPooling* reduz a dimensionalidade do *input*, sendo que, de acordo com o pool size definido, a camada analisa partes de 2x2 *pixels* da imagem de *input* e apenas passa por *output* o valor mais elevado dos quatro *pixels*. Esta camada é bastante eficaz para extrair as *features* mais salientes, e o *pool size* pequeno é indicado para extrair o máximo dessas mesmas *features*.

Após a mesma, existe a última camada Convolucional do modelo, com trinta e dois neurónios, um *filter size* de 3, o *padding* com o valor '*same'* e também a ReLU como função de ativação. Esta camada foi adicionada para extrair as características mais importantes do *input*, dado que o mesmo já sofreu várias alterações desde que esta camada foi utilizada.

Terminando, foi utilizada uma camada *Flatten*. Esta camada transforma os dados num *array* com apenas uma dimensão. O surgimento desta camada representa o começo do fim da rede neuronal, dado que, ao transformar os dados em valores de apenas uma dimensão, operações como as realizadas nas camadas de *MaxPooling* e *Convolutional2D* não poderão ser mais aplicadas. No entanto, ao transformar os dados num *array* de uma dimensão (1D), permite que o modelo possa realizar operações específicas que preparem os dados para a camada final.

Após isso, existe uma camada *Dense* com 64 neurónios e a ReLU como função de ativação uma última camada de *BatchNormalization*, seguida de uma camada de *Dropout* também com 0.6 de *dropout value*. Já a última camada do modelo, apresenta uma camada *Dense* com 100 neurónios com a *softmax* como função de ativação. O número de neurónios deve-se a ser esse o número de classes existentes após a normalização dos dados.

Ao longo do modelo, a *ReLU* foi a função de ativação de eleição por introduzir não-linearidade ao modelo, ou seja, capacitar o mesmo de ser capaz de capturar padrões complexos que não podem ser representados por uma função linear e promover o *sparsity* que faz com que uma certa porção dos neurónios do modelo inativos, aumentando a eficiência computacional e reduzindo o *overfitting*. A *ReLU* apenas foi substituída na última camada, onde a função de ativação utilizada foi a *softmax*. Esta função é, indiscutivelmente, a mais indicada para problemas de classificação de várias classes, pois, realiza uma distribuição de probabilidades ao longo dos neurónios da própria camada, ou seja, atribui a cada neurónio, a probabilidade de o input pertencer a uma certa classe [18].

De seguida, apresenta-se o código do modelo (Figura 15) e uma representação gráfica do mesmo (Figura 16).

```
model = models.Sequential()
model.add(layers.Conv2D(16, (2,2), activation='relu', input_shape=(100, 100, 4)))
model.add(layers.Conv2D(16, 3, strides=1, padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.6))
model.add(layers.MaxPooling2D(pool_size=(2,2)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(32, 3, padding='same', activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.6))
model.add(layers.Dense(100, activation='softmax'))
```

Figura 15 - Código da arquitetura da rede neuronal

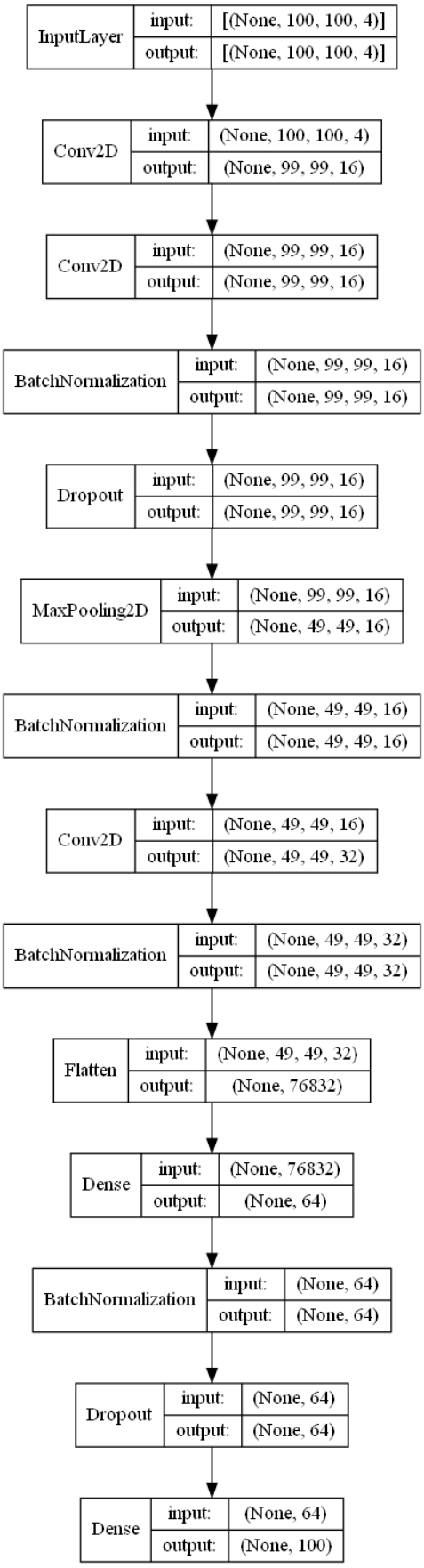

Figura 16 - Representação gráfica da rede neuronal

3.3.2. Parâmetros extra

Para este problema em específico, foram testados vários *optimizers* e *loss functions* mas os escolhidos no final, por apresentarem os melhores resultados, foram o *adam* [19] e a *sparse categorical crossentropy* [20], respetivamente. Foi ainda escolhido a *accuracy* como métrica principal ao compilar o modelo.

A accuracy é, frequentemente, a métrica escolhida para uma CNN por várias razões. Primeiramente, oferece uma interpretação intuitiva sobre a performance da rede neuronal ajudando bastante ao decidir se uma alteração na arquitetura do modelo melhorou ou não a performance do mesmo. A sua versatilidade e aplicabilidade é útil para avaliar diferentes modelos devido ao facto de considerar tanto os acertos quanto os erros do modelo, fornecendo uma avaliação abrangente do desempenho geral do mesmo [21].

Foram ainda desenvolvidas várias funções *callback* a serem utilizadas durante o *fit*. Primeiramente o *early stopping*, que monitorizando a métrica *val_loss* a partir do *epoch* 30, caso não haja uma melhoria no valor dessa mesma métrica durante 15 ciclos seguidos parará a execução. De seguida o *model checkpoint* que, monitorizando a métrica val_*accuracy*, guarda os pesos de cada neurónio num ficheiro específico quando esta métrica atinge o seu valor mais alto registado. Finalmente, como terceira função, existe o *TerminateOnNaN* que não recebe nenhum parâmetro de entrada, apenas termina a execução quando um valor não numérico é encontrado [22].

3.4. Treino do modelo

3.4.1. Primeira abordagem

Primeiramente, já que seria impossível a máquina ter em memória mais de vinte e duas mil imagens, foram selecionadas aleatoriamente trezentas e vinte imagens de cada *label* (Tabela 2), com o seu tamanho original de duzentos por duzentos *pixels*, com um *test size* de 45%.

Tabela 2 - Quantidade de dados a ser usada na primeira abordagem.

Quantidade de imagens em que o *label* é 0	Quantidade de imagens em que o *label* é 1	Quantidade de imagens em que o *label* é 2	Quantidade de imagens em que o *label* é 3	Quantidade de imagens em que o *label* é 5
320	320	168	16	8

De seguida, com um *batch size* de 32 foi realizado um treino onde os seus resultados (Tabela 3) foram os seguintes.

Tabela 3 - Resultados da primeira abordagem.

Loss	*Accuracy*	*Validation loss*	*Validation accuracy*
4.3722	0.2713	4.2314	0.3760

Com esses resultados, foi assumido que o problema estaria na discrepância na quantidade de dados de cada valor passando assim, para uma abordagem mais balanceada nesse aspeto.

3.4.2. Segunda abordagem

Para esta abordagem foi tentado usar a mesma quantidade de imagens para cada valor possível. Para fazer isso de forma que fosse facilmente alterável foi criado um conjunto de variáveis.

Foram criados dois dicionários com chaves de zero a cem, onde, o dicionário denominado de *counter dictionary*, tem todos os seus valores a zero e o dicionário denominado de *limit dictionary* tem todos os seus valores com a quantidade máxima de cada valor definida pelo próprio *developer*.

Por exemplo, se o *developer* entender que deverão ser usadas apenas 20 imagens para treino, nenhuma classe terá mais imagens do que esse valor. Desta forma foi possível equilibrar o *dataset* de treino para obter uma melhor precisão.

Ao ler uma imagem do *dataset*, é verificado antes qual é a label associada à imagem, e, usando esse valor como chave para ambos os dicionários mencionados anteriormente, a imagem apenas é adicionada à lista de dados para treino se, o valor no *counter dictionary* para essa chave for inferior ao valor do *limit dictionary* para essa mesma chave.

Para esta abordagem foram usadas apenas vinte imagens de cada valor, sendo a quantidade de valores a seguinte (Tabela 4).

Tabela 4 - Quantidade de dados a ser usada na segunda abordagem.

Quantidade de imagens em que o *label* é 0	Quantidade de imagens em que o *label* é 1	Quantidade de imagens em que o *label* é 2	Quantidade de imagens em que o *label* é 3	Quantidade de imagens em que o *label* é 5
20	20	20	16	8

De seguida, foi realizado um treino onde os seus resultados foram os seguintes (Tabela 5).

Tabela 5 - Resultados da segunda abordagem.

Loss	*Accuracy*	*Validation loss*	*Validation accuracy*
0.1181	1.0000	2.8125	0.2895

Sendo que os resultados não foram promissores, a abordagem tomada foi reduzir o tamanho das imagens para 100×100 *pixels*.

3.4.3. Terceira abordagem

Para esta abordagem, foi criada uma função (Figura 17) que reduz o tamanho da imagem no momento da adição da mesma à lista de imagens para treino [23].

```
def resize_image(image_path):
    # Open the image file
    image = Image.open(image_path)

    # Resize the image to 100x100 pixels
    resized_image = image.resize((100, 100))

    # Return the resized image
    return resized_image
```

Figura 17 - Código da função de redimensionamento da imagem

Para esta abordagem foram usadas apenas trinta e duas imagens de cada valor, sendo a quantidade de valores a seguinte (Tabela 6).

Tabela 6 - Quantidade de dados a ser usada na terceira abordagem.

Quantidade de imagens em que o *label* é 0	Quantidade de imagens em que a o *label* é 1	Quantidade de imagens em que o *label* é 2	Quantidade de imagens em que o *label* é 3	Quantidade de imagens em que o *label* é 5
32	32	32	16	8

De seguida, foi realizado um treino onde os seus resultados foram os seguintes (Tabela 7).

Tabela 7 - Resultados da terceira abordagem.

Loss	*Accuracy*	*Validation loss*	*Validation accuracy*
0.0290	1.0000	0.2586	1.0000

3.4.4. Abordagem Final

Sendo a terceira abordagem um sucesso, a abordagem final foi alterar os dados de forma que pudesse ser feita uma previsão em relação ao futuro, para isso, foram feitos três treinos a mais.

De forma a ser possível fazer uma previsão ao ser adicionada uma imagem para treino a sua *label* foi o valor da precipitação uma, duas e três horas após.

De forma a realizar treinos eficazes dada a grande diferença de quantidade de valores, para cada hora, primeiro foi realizado um treino apenas com cento e sessenta imagens cuja suas classes fossem 0, 1 ou 2, os três níveis de precipitação mais comuns no *dataset*, e com um *test size* de 40%. Ainda para evitar a repetição das mesmas imagens para treino dado que, no caso destas classes, a quantidade de valores era alta, ao recolher os dados para treino a ordem de obtenção dos dados por *dataset*, seja o original ou os que sofreram *data augmentation*, e por id de estação era alterado a cada utilização.

Após isso foram utilizadas a maior quantidade de imagens possível de modo que houvesse um *dataset* balanceado com todos os valores normalizados obtidos.

Ao fazer isto, a rede neuronal artificial pode entender melhor as características das classes mais abundantes e ainda assim ter capacidade para classificar as imagens das classes mais escassas.

Com uma hora de diferença, os resultados foram os seguintes (Tabela 8).

Tabela 8 - Resultados da abordagem final para uma hora de diferença entre a imagem e a label

	Loss	*Accuracy*	*Validation loss*	*Validation accuracy*
Imagens cuja classe é o valor 0,1 e 2	0.2747	0.9231	0.6504	0.9048
Com todas as classes	1.2858	0.7273	0.8015	0.8889

Já para o treino com duas horas de diferença, os resultados foram os seguintes (Tabela 9).

Tabela 9 - Resultados da abordagem final para duas horas de diferença entre a imagem e a label

	Loss	*Accuracy*	*Validation loss*	*Validation accuracy*
Imagens cuja classe é o valor 0,1 e 2	0.0017	1.0000	0.6732	0.8750
Com todas as classes	0.0536	1.0000	1.0226	0.8750

Finalmente, para o treino com três horas de diferença, os resultados foram os seguintes (Tabela 10).

Tabela 10 - Resultados da abordagem final para três horas de diferença entre a imagem e a label

	Loss	*Accuracy*	*Validation loss*	*Validation accuracy*
Imagens cuja classe é o valor 0,1 e 2	0.1037	0.9655	0.4542	0.9138
Com todas as classes	0.7306	0.7500	0.6122	0.9500

No final, a versão final para obter os dados para treino foi a seguinte (Figura 18).

```python
def list_dataset_folders():
    folders = []
    folder_path = os.getcwd()
    folder_path = os.path.join(folder_path, "datasets")
    for name in os.listdir(folder_path):
        if os.path.isdir(os.path.join(folder_path, name)):
            if name.endswith("dataset"):
                folders.append(name)
    return folders

ids = np.array([1240610, 1210718, 1210702, 1200562, 6212124, 1200575, 1200570, 1200571, 1240903, 1210734, 1210707, 1200558, 1200554, 1210683, 7240919, 1210770, 1240566, 1240675])
specific_datasets = list_dataset_folders()

counter_dictionary = {key: 0 for key in range(101)}
limit_dictionary = {key: 20 for key in range(101)} #para usar mais imagens de chuva, aumentar o valor do limite para cada categoria

time_difference = 1
image_size = 100

data_array = np.empty(0)
images_array = np.empty((0, image_size, image_size, 4))

np.random.shuffle(specific_datasets)
np.random.shuffle(ids)

for dataset in specific_datasets:
    for stationID in ids:
    #json
        currentDir = 'datasets/'+dataset+'/precipitation/'+str(stationID)+'.json'
        with open(currentDir) as f:
            # Load the JSON data
            data = json.load(f)
        f.close()
        for date, hours in data.items():
            for hour in hours:

                hora_da_imagem = datetime.strptime(hour, "%H:%M")
                # Subtract one hour
                hora_da_imagem = hora_da_imagem - timedelta(hours=time_difference)
                # Convert back to string
                hora_da_imagem = hora_da_imagem.strftime("%H:%M")
                variable = date + 'T' + hora_da_imagem.replace(':', '') + '.png'
                value = data[date][hour]

                if not os.path.exists('datasets/'+dataset+'/images/'+str(stationID)+'/'+date+'/'+variable):
                    continue  # Skip the current iteration if the image doesn't exist
                if counter_dictionary[value] < limit_dictionary[value]:
                    # if value > 2:         #para usar apenas imagens das calsses 0, 1 e 2 tirar o comentario destas linhas
                    #     continue
                    # else:
                    counter_dictionary[value] += 1
                    data_array = np.append(data_array, value)
                    image = resize_image('datasets/'+dataset+'/images/'+str(stationID)+'/'+date+'/'+variable)
                    img_np = np.array(image)
                    image.close()
                    images_array = np.append(images_array,[img_np], axis=0)
```

Figura 18 - Função para obtenção de dados para treino

4. *Website* desenvolvido para aplicar o modelo

No website desenvolvido, para que fosse possível a aplicação do modelo de aprendizagem, foi necessário, numa primeira fase, desenvolver uma API em *Flask* para enviar os dados para o *frontend* a cada 10 minutos com as previsões climatéricas de Portugal, que fornecem estimativas de precipitação a cada estação mais perto do centro de cada distrito de Portugal. A API desenvolvida em *Flask* utiliza o mesmo modelo criado e treinado anteriormente, utilizando os pesos guardados dos melhores testes realizados na fase de treino.

4.1. API em *Flask*

Para poder apresentar os resultados da previsão do modelo no website, foi fundamental o desenvolvimento de uma API em *Flask*. Esta API teve como ponto de partida a receção da imagem mais recente, fornecida pelos radares do IPMA, através de *web scrapping* realizado no *website* da própria organização. Para isso, foi necessário construir um URL com a data e hora atualizadas, seguindo um formato específico.

O processo para a obtenção da imagem (Figura 19) foi a seguinte:

1. Obter a data e hora em *UTC*, usando a função *datetime.utcnow*.
2. Arredondar a data e hora para o múltiplo de 5 minutos mais próximo, subtraindo o resto da divisão por 5.

$$f(x) = x - (x\ mod 5)$$

3. Subtrair mais 10 minutos, para garantir que a imagem estivesse sempre disponível na API do IPMA.
4. Formatar a data e hora de acordo com o padrão "%Y-%m-%dT%H%M"[1].
5. Construir o URL para fazer o pedido *Hypertext Transfer Protocol* (HTTP) à API do IPMA, usando uma parte fixa e uma parte variável (data e hora):

   "https://www.ipma.pt/resources.www/transf/radar/por/pcr-%Y-%m-%dT%H%M[2].png"
6. Obter a imagem dos radares de Portugal com o conteúdo da resposta do pedido HTTP.

[1] %Y – ano; %m – mês; %d – dia; %H – hora; %M – minuto;
[2] %Y – ano; %m – mês; %d – dia; %H – hora; %M – minuto;

7. Retornar uma mensagem de erro no formato *JavaScript Object Notation* (JSON), no caso de ocorrer algum erro durante o pedido HTTP.

```python
def get_radar_image():
    try:
        # Obter a hora atual em UTC
        current_datetime_utc = datetime.datetime.utcnow()

        # Arredondar a hora atual para o múltiplo de 5 mais próximo
        rounded_datetime = current_datetime_utc - datetime.timedelta(minutes=current_datetime_utc.minute % 5)

        # Atrasar a hora em 10 minutos
        datetime_delay = rounded_datetime - datetime.timedelta(minutes=10)

        # Formatar a hora atrasada no padrão desejado para criar a URL
        target_datetime_str = datetime_delay.strftime("%Y-%m-%dT%H%M")

        # Construir o URL com a data e hora atualizada
        image_url = f"https://www.ipma.pt/resources.www/transf/radar/por/pcr-{target_datetime_str}.png"

        # Obter a imagem do radar
        response = requests.get(image_url)
        image_data = io.BytesIO(response.content)
        image = Image.open(image_data)

        # Retornar a imagem
        return image

    except Exception as e:
        print(e)
        return jsonify({'error': 'Ocorreu um erro'}), 500
```

Figura 19 – Código da função *get_radar_image*().

A imagem obtida será então usada no *endpoint process_image* que recebe um pedido *GET* e retorna um objeto no formato JSON com as previsões da precipitação para cada estação meteorológica e para cada hora de diferença.

O *endpoint* da previsão de precipitação (Figura 20) faz o seguinte processo:

1. Obter a imagem atual dos radares a partir da função *get_radar_image*.
2. Verificar se a imagem foi obtida com sucesso. Se não, retornar uma mensagem de erro.
3. Criar um dicionário onde as *keys* são as horas de previsão (1, 2 e 3).
4. Obter a lista dos ficheiros de pesos do modelo que estão guardadas na pasta "*model_weights_by_hour*" com a ajuda da função *listdir*.
5. Para cada id na lista de ids das estações, fazer as seguintes operações:
    a. Recortar a imagem do radar com base nas coordenadas fornecidas pelo dicionário *station_box_dict* com a ajuda da função *crop* do módulo *PIL*.
    b. Redimensionar a imagem recortada pela função *resize_image*.
    c. Converter a imagem para um array com a função *img_to_array*

d. Normalizar o *array* da imagem com a função normalize do *TensorFlow*, que recebe o *array* e o eixo com argumentos e retorna um novo *array* normalizado.

e. Fazer a previsão com o valor desnormalizado da precipitação obtido pela função *desnormalize_precipitation_value*.

6. Adicionar a previsão ao dicionário
7. Retornar o dicionário completo no formato JSON.

```python
@app.route('/process_image', methods=['GET'])
def process_image():

    current_radar_image = get_radar_image()

    if current_radar_image is None:
        return 'Falha ao obter a imagem do radar.'

    # Create a dictionary where the keys are the prevision hour
    full_hour_prediction_dictionary = {1: None, 2: None, 3: None}

    model_weights_list = file_list = os.listdir("model_weights_by_hour/")

    for model_weights in model_weights_list:
        try:
            model.load_weights("model_weights_by_hour/"+str(model_weights))

        except Exception as e:
            print("Erro ao carregar os pesos do modelo: ", e)

        finally:
            try:
                current_hour_prediction_dictionary = {}
                for id in ids:

                    # Recortar a imagem com base nas coordenadas fornecidas
                    cropped_image = current_radar_image.crop(station_box_dict[id])

                    # Redimensionar a imagem
                    cropped_image = resize_image(cropped_image)

                    # Converter a imagem para um array
                    array_image = np.array([np.array(cropped_image)])

                    # Normalizar a imagem
                    normalized_array_image = normalize(array_image, axis=1)

                    # Previsão usando o modelo carregado
                    predictions = model.predict(normalized_array_image)

                    #Adicionar a previsão ao dicionário
                    current_hour_prediction_dictionary[station_true_name_dict[id]] = denormalize_precipitation_value(int(np.argmax(predictions[0])))

                    #print()
            except Exception as e:
                print(e)
                return 'Erro ao processar a imagem.'

            finally:
                full_hour_prediction_dictionary[str(extract_number_from_filename(model_weights))] = current_hour_prediction_dictionary
    return json.dumps(full_hour_prediction_dictionary)
```

Figura 20 - Código do endpoint *process_image*().

4.2. *Web Meteo*

O *website* desenvolvido (Figura 21) permite ao utilizador aceder aos resultados das previsões da precipitação em Portugal, em cada distrito logo quando o *website* é carregado. A data e hora da previsão é a atual, mas com 10 minutos de atraso e arredondada por múltiplos de 5. Os resultados da previsão aparecem todos numa única tabela.

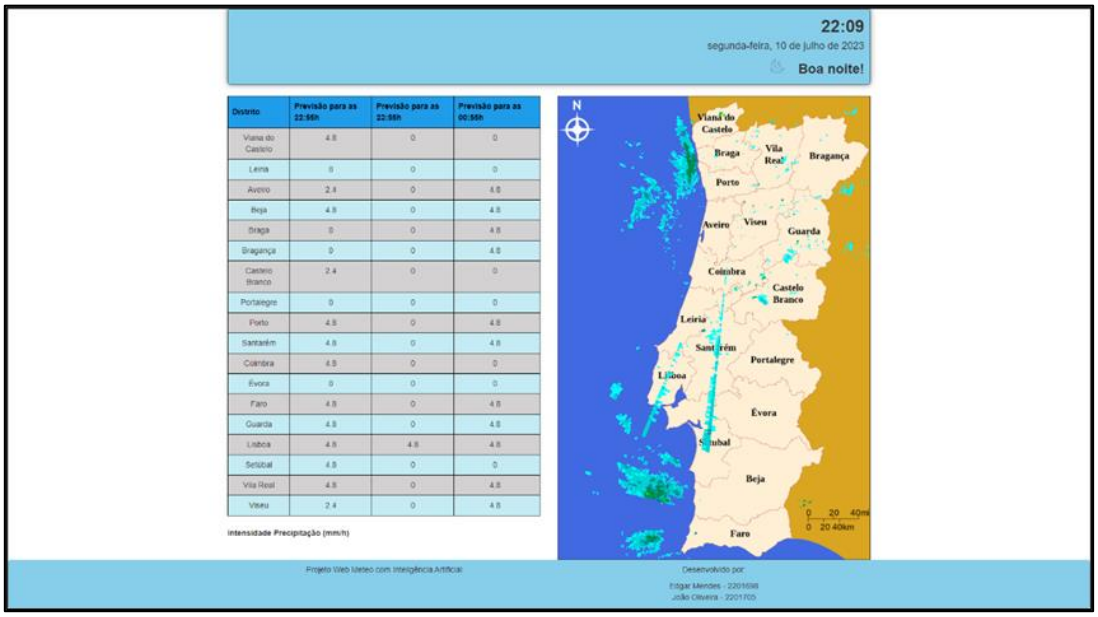

Figura 21 - *Website Web Meteo.*

4.2.1. Implementação

O *website* consiste num ficheiro principal *app.vue* e outros três componentes: *Clock.vue*, *ImagePrediction.vue* e *Mapa.vue*.

O *app.vue* (Figura 22) é responsável por criar uma *navbar*, onde apresenta o relógio, a data e uma saudação de acordo com a hora do dia, importado a partir do componente *Clock.vue*. Também importa o *ImagePrediction.vue* e o *Mapa.vue* que os coloca no conteúdo do website em 2 colunas separadas. Além disso, o *app.vue* define um intervalo de tempo de 5 minutos para chamar as funções principais do *ImagePrediction.vue* e do *Mapa.vue*, que são responsáveis por atualizar os dados da previsão meteorológica e da imagem do radar, respetivamente (Figura 23). O *app.vue* também cria um rodapé com o título e os autores do projeto.

```
<template>
  <!-- Criar uma navbar com bootstrap -->
  <div class="container">
    <div class="row mt-3">
      <div class="col-md-12 text-center">
        <!-- <clock /> -->
        <clock ref="clock" />
      </div>
    </div>

    <div class="row">
      <div class="col-md-12">
        <div class="row">
          <div class="col-md-6">
            <image-predictions ref="imagePredictions" />
          </div>
          <div class="col-md-6">
            <Mapa ref="mapa" />
          </div>

        </div>
      </div>
    </div>
  </div>

  <footer class="text-center text-lg-start">
    <!-- Grid container -->
    <div class="container p-4">
      <!--Grid row-->
      <div class="row">
        <!--Grid column-->
        <div class="col-lg-6 col-md-12 mb-4 mb-md-0">
          <h5>Projeto Web Meteo com Inteligência Artificial</h5>
        </div>
        <!--Grid column-->

        <!--Grid column-->
        <div class="col-lg-6 col-md-12 mb-4 mb-md-0">
          <h5>Desenvolvido por:</h5>
          <p>
            Edgar Mendes - 2201698
            <br />
            João Oliveira - 2201705
          </p>
        </div>
        <!--Grid column-->
      </div>
      <!--Grid row-->
    </div>
    <!-- Grid container -->
  </footer>
</template>
```

Figura 22 - Código do *template* do *app.vue*.

```vue
<script>
import Clock from "./components/Clock.vue";
import Mapa from "./components/Mapa.vue";
import ImagePredictions from "./components/ImagePredictions.vue";

export default {
  name: "App",
  components: {
    Clock,
    Mapa,
    ImagePredictions,
  },
  mounted() {
    setInterval(() => {
      this.$refs.imagePredictions.fetchData();
      this.$refs.mapa.getRadarImage();
    }, 300000);
  },
};

</script>
```

Figura 23 - Código do *script* do *app.vue*.

O *Clock.vue* é um componente que mostra o relógio, a data e uma saudação de acordo com a hora do dia (Figura 24). O componente usa o objeto *Date* do *JavaScript* para obter a hora e a data atuais, e as formata usando o método *toLocateString* com a opção 'pt-PT'. O componente também usa uma condição para determinar se é de manhã, tarde ou noite, e mostra uma imagem do sol ou da lua e uma saudação correspondente. O componente atualiza o relógio a cada minuto usando o método *setInterval* (Figura 25).

```vue
<template>
  <div class="clock-container col-md-12"> <!-- mudar para col-md-8 offset-md-3 quando conseguir ce
    <div class="clock-time">{{ formattedTime }}</div>
    <div class="clock-date">{{ formattedDate }}</div>
    <div>
      <div class="greeting">
        <img :src="iconPath" :alt="iconAlt" class="daytime-icon" :style="{ filter: iconFilter }">
        <span>{{ greeting }}</span>
      </div>
    </div>
  </div>
</template>
```

Figura 24 - Código do *template* do *Clock.vue*.

```vue
<script>
export default {
  data() {
    return {
      formattedTime: '',
      formattedDate: '',
      greeting: '',
      iconPath: '',
      iconAlt: ''
    };
  },
  mounted() {
    this.updateTime();
    setInterval(this.updateTime, 60000); // Atualiza a cada minuto (60 * 1000 ms)
  },
  methods: {

    updateTime() {
      const now = new Date();

      const timeOptions = {
        hour: 'numeric',
        minute: 'numeric'
      };
      this.formattedTime = now.toLocaleString('pt-PT', timeOptions);

      const dateOptions = {
        weekday: 'long',
        day: 'numeric',
        month: 'long',
        year: 'numeric'
      };
      this.formattedDate = now.toLocaleString('pt-PT', dateOptions);

      const hour = now.getHours();
      if (hour >= 5 && hour < 12) {
        this.greeting = 'Bom dia!';
        this.iconPath = '/sun.png'; // Caminho relativo à pasta "public"
        this.iconAlt = 'Ícone do Sol';
      } else if (hour >= 12 && hour < 18) {
        this.greeting = 'Boa tarde!';
        this.iconPath = '/sun.png'; // Caminho relativo à pasta "public"
        this.iconAlt = 'Ícone do Sol';
      } else {
        this.greeting = 'Boa noite!';
        this.iconPath = '/moon.png'; // Caminho relativo à pasta "public"
        this.iconAlt = 'Ícone da Lua';
      }
    }

  }
};
</script>
```

Figura 25 - Código do *script* do *Clock.vue*.

O *ImagePredictions.vue* é um componente que mostra uma tabela com as previsões da precipitação para os distritos de Portugal, baseadas numa imagem do radar (Figura 26). O componente usa o módulo *axios* para fazer um pedido *GET* à API *Flask*, que é responsável por processar a imagem do radar e retornar um objeto JSON com os dados das previsões. O componente então formata os dados recebidos num objeto que pode ser usado para preencher a tabela, usando um *loop v-for*. O componente também mostra a hora em que as previsões são feitas, que é obtida usando o objeto *Date* do *JavaScript* e arredondada para o múltiplo de 5 minutos mais próximo subtraindo mais 10 minutos (Figura 27).

```
<template>
  <div class="dataTable">
    <div class="table-responsive">
      <table class="table table-bordered text-center">
        <thead>
          <tr>
            <th>Distrito</th>
            <th>Previsão para as {{ time_delay1h }}h</th>
            <th>Previsão para as {{ time_delay2h }}h</th>
            <th>Previsão para as {{ time_delay3h }}h</th>
          </tr>
        </thead>
        <tbody>
          <tr v-for="(item, key) in formattedData" :key="key">
            <td>{{ key }}</td>
            <td>{{ item[0] }}</td>
            <td>{{ item[1] }}</td>
            <td>{{ item[2] }}</td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
  <div v-if="isLoading" class="text-center">
    <p>A calcular previsões...</p>
  </div>
    <p style="font-weight: bold">Intensidade Precipitação (mm/h)</p>
</template>
```

Figura 26 - Código do *template* do *ImagePrediction.vue*.

```vue
<script>
import axios from "axios";

export default {
  data() {
    return {
      formattedData: {},
      isLoading: true,
      time_delay1h: "",
      time_delay2h: "",
      time_delay3h: "",
    };
  },
  mounted() {
    this.fetchData();
    // setInterval(this.fetchData, 300000); // 5 minutes
  },
  methods: {
    fetchData() {
      this.isLoading = true;
      this.formattedData = {};
      this.time_delay1h = "";
      this.time_delay2h = "";
      this.time_delay3h = "";
      console.log("Fetching data...");

      axios
        .get("http://localhost:5000/process_image")
        .then((response) => {
          const data = response.data;
          for (const [key, value] of Object.entries(data)) {
            for (const [key2, value2] of Object.entries(value)) {
              if (!this.formattedData[key2]) {
                this.formattedData[key2] = [];
              }
              this.formattedData[key2].push(value2);
            }
          }
          this.isLoading = false;
          console.log(this.formattedData);

          // Obter a hora atual em UTC
          var current_datetime_utc = new Date();
          // Arredondar a hora atual para o múltiplo de 5 mais próximo
          var rounded_datetime = new Date(current_datetime_utc);
          rounded_datetime.setMinutes(
            Math.floor(rounded_datetime.getMinutes() / 5) * 5
          );
          // Atrasar a hora em 10 minutos
          var datetime_delay = new Date(
            rounded_datetime.getTime() - 10 * 60000 + 3600000
          );
          // Converter a hora para o formato hh:mm
          this.time_delay1h = datetime_delay.toLocaleTimeString([], {
            hour: "2-digit",
            minute: "2-digit",
          });
          // console.log(this.time_delay1h);
          this.time_delay2h = new Date(
            datetime_delay.getTime() + 3600000
          ).toLocaleTimeString([], {
            hour: "2-digit",
            minute: "2-digit",
          });
          // console.log(this.time_delay2h);
          this.time_delay3h = new Date(
            datetime_delay.getTime() + 2 * 3600000
          ).toLocaleTimeString([], {
            hour: "2-digit",
            minute: "2-digit",
          });
          // console.log(this.time_delay3h);
        })
        .catch((error) => {
          console.error(error);
        });
    },
  },
};
</script>
```

Figura 27 - Código do *script* do *ImagePredictions.vue*.

O *Mapa.vue* é um componente que mostra uma imagem do mapa de Portugal com a imagem do radar encaixada pelos limites da imagem do mapa e sobreposta (Figura 28). O componente usa o módulo *axios* para fazer um pedido *GET* à API *Flask*, que é responsável por obter a imagem do radar mais recente e retorná-la em formato *base64* (Figura 29). O componente então usa o atributo *src* da *tag img* para mostrar a imagem do radar, usando uma posição absoluta e um tamanho ajustado para se sobrepor ao mapa de Portugal, que é uma imagem estática na pasta *assets* (Figura 30).

```vue
<template>
  <div class="home-container">
    <!--insert image-->
    <div class="map-container">
      <div class="radar-container">
        <img :src="radarImage" alt="Mapa de Portugal" class="radar-image" />
      </div>
      <img
        src="../assets/mapaPortugal.png"
        alt="Mapa de Portugal"
        class="image"
      />
    </div>
  </div>
</template>
```

Figura 28 - Código do *template* do Mapa.vue.

```vue
<script>
import axios from "axios";

export default {
  name: "Mapa de Portugal",
  data() {
    return {
      radarImage: null,
    };
  },
  async mounted() {
    this.getRadarImage();
    // setInterval(this.getRadarImage, 300000); // 5 minutes
  },
  methods: {
    getRadarImage() {
      console.log("Getting radar image...");
      this.radarImage = null;
      axios
        .get("http://localhost:5000/radar-image")
        .then((response) => {
          const data = response.data;
          this.radarImage = `${data}`;
          console.log("Radar image received!");
        })
        .catch((error) => {
          console.log(error);
        });
    },
  },
};
</script>
```

Figura 29 - Código do *script* do Mapa.vue.

```
<style scoped>
1 reference
.home-container {
  height: auto;
  width: auto;
  display: flex;
  align-items: center;
  border-color: var(--dl-color-gray-black);
  border-style: solid;
  border-width: 1px;
  background-color: #87ceeb;
  flex-direction: row;
  justify-content: center;
  vertical-align: middle;
}

1 reference
.image {
  width: 100%;
  height: 100%;
  object-fit: cover;
}

1 reference
.map-container {
  position: relative;
  width: 100%;
  height: 100%;
  overflow: hidden;
}

1 reference
.radar-image {
  position: absolute;
  top: -32%;
  left: -37%;
  width: 180%;
  height: 184%;
}
1 reference
.radar-container {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
}
</style>
```

Figura 30 - Código css do Mapa.vue.

4.3. Instalação e uso

O projeto *WebMeteo* consiste numa aplicação *web* que mostra as previsões da precipitação em cada estação referente a cada distrito de Portugal após 1, 2 e 3 horas. A aplicação *web* é feita com *vue.js* e os dados são fornecidos por uma API feita com *Flask*. As instruções seguintes explicam como executar o projeto.

4.3.1. Requisitos

Antes de executar o projeto, é preciso ter instalado os seguintes programas:

- *Pyhton*
- *Flask*
- *Node.js*
- *Npm*

4.3.2. Passos

Para executar o projeto é necessário realizar os seguintes passos:

1. Num terminal a diretoria deverá ser alterada para a seguinte pasta `WebMeteoPI/FlaskFramework`.
2. Será também necessário instalar as seguintes bibliotecas para o código *Flask* funcionar corretamente, usando os seguintes comandos:
   - `pip install flask`
   - `pip install tensorflow`
   - `pip install Pillow`
   - `pip intall flask-cors`
3. Após isso será necessário executar a API, usando o comando `py app.py run`.
4. Noutro terminal a diretoria deverá ser alterada para a seguinte pasta `WebMeteoPI/vue-client`.
5. Será também necessário instalar as dependências necessárias para o *vue.js*, usando o comando `npm install`.
6. Para executar o *website* deverá ser usado o comando `npm run dev`.
7. Já no *browser*, para ver a aplicação *web* a funcionar, é necessário aceder a este link http://localhost:5173/

5. Análise de resultados

A capacidade de uma rede neuronal identificar algo é uma coisa que tem recebido muita atenção e esforço por parte de toda a gente que se dedica ao ramo da inteligência artificial. Ferramentas presentes no nosso dia-a-dia como, reconhecimento facial, que se encontra na maioria dos telemóveis atuais, reconhecimento de voz, entre muitos outros exemplos têm todos como base as redes neuronais.

Estas redes ou modelos podem ser avaliadas de várias formas, sendo que, neste projeto, a métrica considerada mais importante, foi a *validation accuracy*. O valor desta métrica é a taxa de acerto do modelo para imagens nunca vistas pelo modelo, mas que estão devidamente classificadas. Com as ferramentas que utilizámos o valor desta métrica está compreendido entre 0 e 1 tendo de ser multiplicado por 100 para apresentar o valor em percentagem.

Mesmo assim, também foi dada atenção a outras métricas sendo elas também importantes. Primeiramente a *accuracy*, que se traduz na taxa de acerto para as imagens com que o próprio modelo foi treinado, sendo assim, o valor da *accuracy* pode às vezes dar resultados muito bons, mas, na prática, o modelo pode não ter uma boa capacidade de classificação. Isto acontece quando existe um dos problemas mais comuns, no entanto dos mais perigosos para o bom funcionamento de um modelo, o *overfitting*. Este problema acontece quando o modelo 'viu' demasiadas vezes a mesma imagem, então, perde a sua capacidade de generalização, ou seja, ao ver uma imagem da mesma classe, mas que seja significativamente diferente, o modelo não será capaz de classificar corretamente.

Para evitar o *overfitting* ao máximo, dado que, ao longo do projeto, existiu uma grande carência de dados por fatores externos tanto aos alunos quanto aos professores orientadores, durante o processo de treino foram realizadas duas etapas. Na primeira etapa, foi realizado um treino onde foram utilizados apenas dados que existiam em 'grande' quantidade, ou seja, como mencionado na descrição do processo, imagens cujo a sua classificação fosse o valor normalizado de 0, 1 ou 2. Já na segunda e última etapa, foi realizado um treino com imagens de todas as classificações, mas com *dataset* equilibrado, ou seja, utilizando o mesmo número de imagens para cada classe, sendo assim, o treino foi realizado com apenas 8 imagens de cada classe dado que esse era o número total de imagens pertencentes a essa classe. Sendo assim, ainda que possa ter acontecido, inevitavelmente, um certo *overfitting* para as classes mais escassas, para as classes onde existiam bastantes dados foram utilizados uma grande quantidade

de dados diferentes em cada treino sendo que para essas, é considerado que o modelo tem uma boa capacidade de generalização.

Relativamente aos resultados em si, após o treino por etapas e o *data augmentation*, foi possível notar uma melhoria na previsão realizada, sendo que para as previsões das próximas uma, duas e três horas foi obtido um valor de *validation accuracy* de 88,89%, 87,5% e 95% respetivamente.

Ainda assim, o modelo não está preparado para imagens de radar de chuva intensa dado que esses dados não estiveram disponíveis no treino.

6. Conclusão

O propósito deste projeto foi a criação de um *website* que fosse capaz de mostrar a precipitação num futuro próximo, prevista por uma rede neuronal.

Em termos de precisão, o modelo ficou um pouco aquém da expectativa devido à falta de dados pelas limitações do IPMA, e pelo mesmo ter recusado o nosso pedido de fornecer mais. Ainda assim, o projeto tem implementado todas as funções necessárias para esses dados serem obtidos por um futuro utilizador do mesmo.

Mesmo assim, com o desenvolvimento deste projeto foi possível criar uma ferramenta com grande potencial, tendo até capacidades de interpretar detalhes e padrões que não seria possível analisar de outra forma, para esta área que já se provou ser algo que tem uma grande influencia nas decisões de alguém.

Apesar de todos os obstáculos que tivemos ao longo do projeto, tanto os que conseguimos ultrapassar como o que não conseguimos, aprendemos bastante com todos os erros e problemas, podendo dizer que o projeto ficou próximo do seu pico de qualidade, ignorando os problemas externos a nós, tendo ainda, como sempre, espaço para melhorias.

Para implementações futuras, este projeto poderia ser melhorado se fosse implementado uma rede neuronal de *Deep Reinforcement Learning*, o que poderia aumentar a precisão e a eficiência do modelo. Outra forma de melhorar o projeto seria poder apresentar mais dados e imagens reais vindos do IPMA, o que poderia enriquecer o *dataset* utilizado para o treino da inteligência artificial e fornecer mais informação relevante para a previsão da precipitação.

Bibliografia ou Referências Bibliográficas

[1] A. Bonner, “Towards Data Science,” 7 September 2019. [Online]. Available: https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-f7d02aa9d477.

[2] M. H. Buettgenbach, “Towards Data Science,” 8 November 2021. [Online]. Available: https://towardsdatascience.com/explain-like-im-five-artificial-neurons-b7c475b56189.

[3] S. Sharma, “Towards Data Science,” 6 September 2021. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[4] A. Tch, “Towards Data Science,” 4 August 2017. [Online]. Available: https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464.

[5] M. Ross, “Towards Data Science,” 10 September 2017. [Online]. Available: https://towardsdatascience.com/under-the-hood-of-neural-network-forward-propagation-the-dreaded-matrix-multiplication-a5360b33426.

[6] S. Enslin, “Towards Data Science,” 14 August 2021. [Online]. Available: https://towardsdatascience.com/the-complete-guide-to-neural-networks-multinomial-classification-4fe88bde7839.

[7] V. Kaushik, “Analytics Steps,” 21 August 2021. [Online]. Available: https://www.analyticssteps.com/blogs/8-applications-neural-networks.

[8] Prabhu, “Medium,” 4 March 2018. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148.

[9] J. Brownlee, “Machine Learning Mastery,” 17 April 2019. [Online]. Available: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/.

[10] R. Qayyum, “Towards AI,” 16 August 2022. [Online]. Available: https://towardsai.net/p/l/introduction-to-pooling-layers-in-cnn.

[11] D. Unzueta, “Towards Data Science,” 13 November 2021. [Online]. Available: https://towardsdatascience.com/convolutional-layers-vs-fully-connected-layers-364f05ab460b.

[12] “Kinsta,” 13 December 2022. [Online]. Available: https://kinsta.com/knowledgebase/what-is-github/.

[13] B. Pryke, “Data Quest,” 24 August 2020. [Online]. Available: https://www.dataquest.io/blog/jupyter-notebook-tutorial/.

[14] P. Pedamkar, “EDUCBA,” 13 March 2023. [Online]. Available: https://www.educba.com/what-is-visual-studio-code/.

[15] H. Shah, “Able Bio,” 1 December 2021. [Online]. Available: https://able.bio/hardikshah/6-reasons-why-flask-is-better-framework-for-web-application-development--cd398f73.

[16] J. Patel, “Monocubed,” 5 August 2021. [Online]. Available: https://www.monocubed.com/blog/advantages-of-vue-js/.

[17] “NVIDIA,” [Online]. Available: https://www.nvidia.com/en-us/glossary/data-science/tensorflow/.

[18] P. Soni, “Analytics Steps,” 9 January 2022. [Online]. Available: https://www.analyticssteps.com/blogs/data-augmentation-techniques-benefits-and-applications.

[19] U. Jaitley, “Medium,” 7 October 2017. [Online]. Available: https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029.

[20] Y. Upadhyay, “Medium,” 4 January 2019. [Online]. Available: https://medium.com/alumnaiacademy/introduction-to-computer-vision-4fc2a2ba9dc.

[21] D. Giordano, “Towards Data Science,” 25 July 2020. [Online]. Available: https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e.

[22] J. Brownlee, “Machine Learning Mastery,” 30 January 2019. [Online]. Available: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/.

[23] A. Mishra, “Towards Data Science,” 24 February 2018. [Online]. Available: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234.

[24] A. Duong, “KDNuggets,” August 2019. [Online]. Available: https://www.kdnuggets.com/2019/08/keras-callbacks-explained-three-minutes.html.

[25] M. Anderson, “Unite AI,” 9 December 2022. [Online]. Available: https://www.unite.ai/better-machine-learning-performance-through-cnn-based-image-resizing/.

Anexos

Anexo 1 - Relatórios semanais

- **Relatório Semanal nº01** (1/3/2023 18:30h → 7/3/2023 18:30h):
  O trabalho realizado neste intervalo de tempo foi o seguinte:
  - Ambos os membros leram relatório do ano anterior disponibilizado pelos professores orientadores;
  - Ambos os membros com o Jupyter Notebook e o Tensorflow fizeram um algoritmo de reconhecimento de algarismos de 0 a 9 (mnist).
- **Relatório Semanal nº02** (7/3/2023 18:30h → 14/3/2023 21:00h):
  O trabalho realizado neste intervalo de tempo foi o seguinte:
  - Esta semana analisámos, comentámos e otimizámos o código de reconhecimento de voz fornecido pelo professor orientador conseguindo um valor máximo de *val_accuracy* de 0.9655.
- **Relatório Semanal nº03** (14/3/2023 21:00h → 20/3/2023 21:03h):
  O trabalho realizado neste intervalo de tempo foi o seguinte:
  - Esta semana analisámos, comentámos e otimizámos o código de *Deep Reinforcement Learning* fornecido pelos nossos professores orientadores.
- **Relatório Semanal nº04** (20/3/2023 21:03h → 31/3/2023 22:00h):
  O trabalho realizado neste intervalo de tempo foi o seguinte:
  - Lemos vários artigos e relatórios de projetos sobre previsão de tempo, tirando deles algumas ideias para utilizarmos no nosso próprio relatório;
  - Não fomos capazes de realizar o trabalho proposto pelos professores orientadores de procurar código de projetos semelhantes. Tentámos em várias plataformas como o Github e ainda procurámos um Dataset na plataforma Kaggle mas não encontrámos nenhum que se assemelhasse ao que nós procurávamos;
  - De forma a não ficar atrasados no desenvolvimento do projeto, escrevemos um código que vai buscar as imagens dos radares do IPMA e guarda numa pasta cada imagem com a data e a hora.

- **Relatório Semanal nº05** (31/3/2023 22:00h → 11/4/2023 21:30h)

  O trabalho realizado neste intervalo de tempo foi o seguinte:

  - Arranjámos forma de poder organizar o dataset com as imagens e os respetivos valores da precipitação vindas da API do IPMA para todas as estações;
  - Cortámos o mapa para o distrito de Viana do Castelo para começar o treino;
  - Fizemos o primeiro esboço do modelo;
  - Fizemos funções auxiliares para ir buscar os dados e guardar localmente, para normalizar o valor da precipitação de 0 a 100 e para criar as respetivas pastas.

- **Relatório Semanal nº06** (11/4/2023 21:30h → 17/4/2023 21:30h)

  O trabalho realizado neste intervalo de tempo foi o seguinte:

  - Conseguimos resolver o erro de input do modelo, sendo assim já conseguimos começar os testes;
  - Começámos a redigir o resumo/*abstract* e a introdução do relatório.

- **Relatório Semanal nº07** (17/4/2023 21:30h → 26/4/2023 21:30h)

  O trabalho realizado neste intervalo de tempo foi o seguinte:

  - Enviámos email ao IPMA no qual responderam, com interesse, em marcar uma reunião.

- **Relatório Semanal nº08** (26/4/2023 21:30h → 02/05/2023 20:00h)

  O trabalho realizado neste intervalo de tempo foi o seguinte:

  - Fizemos um pouco de trabalho de pesquisa para saber como implementar o modelo no site, chegando à decisão de usar Vue.js para o front-end e Flask para o back-end e ainda aprofundámos um pouco os conhecimentos sobre os mesmos, nomeadamente, Flask por ser algo que não foi lecionado ao longo do curso;
  - Juntámos um mapa de Portugal às imagens vindas do IPMA, para, conseguirmos fazer o corte para os distritos, caso se prove necessário;
  - Fizemos o recorte para o distrito de Leiria.

- **Relatório Semanal nº09** (02/05/2023 20:00h → 08/05/2023 21:30h)

  O trabalho realizado neste intervalo de tempo foi o seguinte:

  - Fizemos o recorte para todos os distritos;
  - Gerámos 12 vezes mais imagens com métodos de *augmentation*;
  - Começámos a fazer uma função que faz com que as imagens para treino tenham a mesma quantidade para cada valor.

- **Relatório Semanal nº10** (08/05/2023 21:30h → 15/05/2023 21:30h)

  O trabalho realizado neste intervalo de tempo foi o seguinte:

  - Começámos o website já integrando o modelo (Vue.js + Flask);
  - Fizemos uma função que vai buscar as imagens de duas horas antes;
  - Fizemos uma função que mete no *dataset* para o modelo o mesmo número de imagens para cada valor (Ex.: se às 12h há 1000 imagens com o valor 0 e 216 imagens com o valor 1 o *dataset* vai ter 432 imagens (216 do valor 0 e 216 do valor 1);
  - Começámos a fazer mais treinos com modelos diferentes, mas a precisão não aumentou muito pela falta de imagens;
  - Tentámos usar a arquitetura EfficientNetB7 e ResNet50, mas ambas demoraram demasiado a correr fazendo com que desistíssemos da ideia.

- **Relatório Semanal nº11** (15/05/2023 21:30h → 23/05/2023 22:00h)

  O trabalho realizado neste intervalo de tempo foi o seguinte:

  - Refizemos o *augmentation* com movimentação de 20 pixels para a direita e esquerda, rotação de 1º e ainda fizemos a junção desses dois métodos;
  - Começámos a fazer a previsão de 1 hora;
  - Alterámos algumas coisas no Website.

- **Relatório Semanal nº12** (23/05/2023 22:00h → 29/05/2023 21:30h)

  O trabalho realizado neste intervalo de tempo foi o seguinte:

  - Usámos mais dois *augmentations*. Criámos novos *datasets* com 20 pixels para baixo e um outro com 20 pixels para baixo e 1 grau de inclinação para aumentar o número de imagens com valores distintos;
  - Usámos os dados de todas as horas para o treino do modelo;
  - Melhorámos o website;
  - Reduzimos as imagens para 100 pixels para o treino.

- **Relatório Semanal nº13** (29/05/2023 21:30h →05/06/2023 21:30h)

  O Trabalho realizado neste intervalo de tempo foi o seguinte:

  - Usámos as imagens com os valores de uma hora posterior para o treino do modelo para a previsão de 1 hora;
  - Melhorámos do website.

Anexo 2 - Código da função de obtenção de imagens e valores do IPMA e variáveis uteis para o bom funcionamento da mesma

```
boxVianaDoCastelo, boxLeiria, boxAveiro, boxBeja, boxBraga, boxBraganca,
boxCasteloBranco, boxPortalegre, boxPorto, boxSantarem, boxCoimbra,
boxEvora, boxFaro, boxGuarda, boxLisboa, boxSetubal, boxVilaReal, boxViseu
= ((570, 428, 770, 628), (574, 902, 774, 1102), (603, 687, 803, 887), (735,
1546, 935, 1746), (645, 463, 845, 663), (953, 401, 1153, 601), (817, 886,
1017, 1086), (829, 1012, 1029, 1212), (611, 562, 811, 762), (597, 1026,
797, 1226), (645, 792, 845, 992), (740, 1183, 940, 1383), (736, 1546, 936,
1746), (859, 712, 1059, 912), (513, 1149, 713, 1349), (559, 1193, 759,
1393), (770, 527, 970, 727), (740, 682, 940, 882))
idVianaDoCastelo, idLeiria, idAveiro, idBeja, idBraga, idBraganca,
idCasteloBranco, idPortalegre, idPorto, idSantarem, idCoimbra, idEvora,
idFaro, idGuarda, idLisboa, idSetubal, idVilaReal, idViseu = 1240610,
1210718, 1210702, 1200562, 6212124, 1200575, 1200570, 1200571, 1240903,
1210734, 1210707, 1200558, 1200554, 1210683, 7240919, 1210770, 1240566,
1240675
ids = np.array([1240610, 1210718, 1210702, 1200562, 6212124, 1200575,
1200570, 1200571, 1240903, 1210734, 1210707, 1200558, 1200554, 1210683,
7240919, 1210770, 1240566, 1240675])
station_box_dict = {idVianaDoCastelo: boxVianaDoCastelo, idLeiria:
boxLeiria, idAveiro: boxAveiro, idBeja: boxBeja, idBraga: boxBraga,
idBraganca: boxBraganca, idCasteloBranco: boxCasteloBranco, idPortalegre:
boxPortalegre, idPorto: boxPorto, idSantarem: boxSantarem, idCoimbra:
boxCoimbra, idEvora: boxEvora, idFaro: boxFaro, idGuarda: boxGuarda,
idLisboa: boxLisboa, idSetubal: boxSetubal, idVilaReal: boxVilaReal,
idViseu: boxViseu}

DADOS_ULTIMAS_3_HORAS = "https://api.ipma.pt/open-
data/observation/meteorology/stations/obs-surface.geojson"
local_tz = pytz.timezone('Europe/Lisbon') #Define the current timezone

def get_images_and_data_from_ipma():

    data = get_data(DADOS_ULTIMAS_3_HORAS)
    specific_datasets = list_dataset_folders()
    # Make the API call
    final_result = {}
    for feature in data['features']:
        f.value+=1
        if feature['properties']['idEstacao']  in ids:
            station_data = feature['properties']
            id_estacao = station_data['idEstacao']
            # Convert the string date/time to a Python datetime object (in
UTC+1)
            date_time_utc = datetime.fromisoformat(station_data['time'])
            date_time =
datetime.fromisoformat(station_data['time']).replace(tzinfo=pytz.utc).astim
ezone(local_tz)
            date_str, hour_str = date_time.strftime('%Y-%m-%d
%H:%M').split()
            precipitation = station_data['precAcumulada']

            if date_str not in final_result:
                final_result[date_str] = {hour_str: precipitation}
            else:
```

```
                final_result[date_str][hour_str] =
normalize_precipitation_value(max(0, precipitation))

            url_image =
f"https://www.ipma.pt/resources.www/transf/radar/por/pcr-
{date_time_utc.strftime('%Y-%m-%d')}T{date_time_utc.strftime('%H%M')}.png"
            response = requests.get(url_image)
            image_data = BytesIO(response.content)
            image = Image.open(image_data)

            # Remove the black pixels from the image
            image = remove_black_pixels(image)

            # Cut the image to the station
            region = image.crop(station_box_dict[id_estacao])
            image.close()
            for specific_dataset in specific_datasets:
                # Verify if the station folder exists and create it if it
doesn't
                if not
os.path.exists(f"datasets/{specific_dataset}/images/{id_estacao}"):

os.makedirs(f"datasets/{specific_dataset}/images/{id_estacao}")

                # Verify if the date folder exists and create it if it
doesn't
                if not
os.path.exists(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}
"):

os.makedirs(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}")

                # Save the image in the correct folder
                elif specific_dataset == 'dataset':

region.save(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}/{d
ate_time.strftime('%Y-%m-%dT%H%M')}.png")
                    original_image_path =
f"datasets/dataset/images/{id_estacao}/{date_str}/{date_time.strftime('%Y-
%m-%dT%H%M')}.png"

                elif specific_dataset == 'left_20px_shifted_dataset':
                    shift_image_left(original_image_path,
20).save(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}/{date
_time.strftime('%Y-%m-%dT%H%M')}.png")
                    left_shifted_image_path =
f"datasets/left_20px_shifted_dataset/images/{id_estacao}/{date_str}/{date_t
ime.strftime('%Y-%m-%dT%H%M')}.png"

                elif specific_dataset == 'right_20px_shifted_dataset':
                    shift_image_right(original_image_path,
20).save(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}/{date
_time.strftime('%Y-%m-%dT%H%M')}.png")
                    right_shifted_image_path =
f"datasets/right_20px_shifted_dataset/images/{id_estacao}/{date_str}/{date_
time.strftime('%Y-%m-%dT%H%M')}.png"

                elif specific_dataset == 'down_20px_shifted_dataset':
                    shift_image_down(original_image_path,
20).save(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}/{date
_time.strftime('%Y-%m-%dT%H%M')}.png")
```

```
                    down_shifted_image_path =
f"datasets/down_20px_shifted_dataset/images/{id_estacao}/{date_str}/{date_t
ime.strftime('%Y-%m-%dT%H%M')}.png"

                elif specific_dataset ==
'rotated_1degree__left_20px_shifted_dataset':
                    rotate_image(left_shifted_image_path,
1).save(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}/{date_
time.strftime('%Y-%m-%dT%H%M')}.png")

                elif specific_dataset ==
'rotated_1degree__right_20px_shifted_dataset':
                    rotate_image(right_shifted_image_path,
1).save(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}/{date_
time.strftime('%Y-%m-%dT%H%M')}.png")

                elif specific_dataset ==
'rotated_1degree__down_20px_shifted_dataset':
                    rotate_image(down_shifted_image_path,
1).save(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}/{date_
time.strftime('%Y-%m-%dT%H%M')}.png")

                elif specific_dataset == 'rotated_1degree_dataset':
                    rotate_image(original_image_path,
1).save(f"datasets/{specific_dataset}/images/{id_estacao}/{date_str}/{date_
time.strftime('%Y-%m-%dT%H%M')}.png")

                # Verify if the JSON file for this station already exists,
if not, create the file
                filename =
f"datasets/{specific_dataset}/precipitation/{id_estacao}.json"
                if not os.path.isfile(filename):
                    with open(filename, 'w') as file:
                        json.dump({}, file)

                # Load the JSON file content to the "precipitation_data"
variable
                with open(filename, 'r') as file:
                    precipitation_data = json.load(file)

                # Add the weather information to the JSON file
                for date in final_result:
                    if date not in precipitation_data:
                        precipitation_data[date] = final_result[date]
                    else:
                        precipitation_data[date].update(final_result[date])

                # Write the updated content to the JSON file
                with open(filename, 'w') as file:
                    json.dump(precipitation_data, file, indent=4)
    print("Dados atualizados com sucesso!")
```